Pictorial Representation of Text: Converting Text to Pictures

Nelson David Ludlow

Ph.D.
Department of Artificial Intelligence
University of Edinburgh
1992

Declaration

I hereby declare that I composed this thesis entirely myself and that it describes my own research.

Nelson D. Ludlow
Edinburgh
28 August, 1992

Abstract

"A picture is worth a thousand words." This saying suggests that an inter–relationship exists between text and pictures. This thesis is the result of an investigation to identify and exploit the inter–relationships between text and pictures. It describes a concept of pictorial representation of text, presents a Text–to–Pictures System which generates a pictorial representation of English sentences, and gives a more detailed look at how the system pictorially represents specific linguistic types of natural language expressions.

Although very little previous work has been done in this area, the relevant work in *text–pictures systems* is summarized. Most of the past work concentrated on pictorially representing the nouns and some spatial prepositions. My work expands the pictorial representation to include temporal expressions, conjunction, relative clauses, quantification, and some verb features.

The thesis also addresses the concepts of using pictorial representation for data fusion of large natural language texts, as well as the problems of ambiguity and vagueness.

The working system to "convert text to pictures" is demonstrated. The system structure, intermediate representation schemes, and the translation process are described. Several types of natural language expressions are examined and the corresponding pictorial representations are shown. Also shown is an application to pictorially represent all of the possible meanings of an ambiguous sentence to allow a non–linguist user to choose the intended meaning.

Using a natural language text processing system that can convert a sentence of text into a logical form (LF) representation, I show what is required to convert the LF representation into a pictorial representation. The process involves identifying the *objects* contained in the LF and representing them by icons. These icons are placed in an imaginary space via a set of constraints. After all of the constraints are determined, the system attempts to solve the generated constraint satisfaction problem. If a solution is found, the icons are drawn with the appropriate coordinates on a graphics display.

Notably, the constraint generation is critical to the system. Each sentence is represented by a large set of pictorial constraints, many of which are generated directly from the LF representation. Two examples of pictorial constraints generated directly from the LF representation are constraints generated from verbs and prepositions. Other pictorial constraints are needed to handle naive physics, to provide missing information, to reduce false implicatures, and to simply provide an aesthetic picture.

Acknowledgments

Several people have helped contribute to this thesis. However, any omissions or mistakes are entirely my own.

I would especially like to thank my two advisors: Elisabet Engdahl and Chris Mellish. They were creative, supportive, insightful and understanding. They were simply the best.

I also thank Dan Howland and John Shaud for arranging for my funding and opportunity to pursue my PhD.

Several people have made comments on drafts, suggested new ideas, gave constructive criticism, or helped with programming issues. They include: Hiyan Alshawi, John Beaven, Alan Black, Dave Carter, Matt Crocker, Mariana Damova, John Lee, Ian Lewin, Suresh Manandhar, Carla Pedro Gomes, Dave Moffat, Luis Pineda, Steve Pulman, Wayne Redenbarger, Antonio Sanfillipo, Donia Scott, Rob Scott, and Karen Spark Jones.

Carla Pedro Gomes deserves a special mention as she encouraged me, challenged me, and made it possible to complete the thesis although I was working away from Edinburgh for many months.

Finally, I want to thank my parents, David and Carol, who taught me how to learn, and to strive for anything that I could dream of.

Contents

List of Figures

Chapter 1

Pictorial Representation of Text

"A picture is worth a thousand words." This saying seems to suggest the hypothesis that an inter–relationship exists between text and pictorial representations. A weather map quickly depicts a large amount of data—the same data can be represented in a textual format, but would not be as succinct. But, can this inter–relationship be described?

This thesis is the result of an investigation to identify and exploit the inter–relationships between text and pictures. It describes a concept of pictorial representation of text, a Text–To–Pictures System which presents a pictorial representation of English sentences, and gives a more detailed look at how the system pictorially represents specific linguistic types of natural language expressions.

A working system to "convert text to pictures" is demonstrated. The system structure, intermediate representation schemes, and the translation process are described. Several types of natural language expressions are examined and the corresponding pictorial representations are shown.

Although very little previous work has been done in this area, the relevant work in *text–pictures systems* is summarized. Most of the past work concentrated on pictorially representing the nouns and some spatial prepositions. This work expands the pictorial representation to include temporal expressions, conjunction, relative clauses, quantification, and some verb features.

The thesis also addresses the concepts of using pictorial representation for data fusion of large natural language texts, as well as the problems of ambiguity and vagueness.

1.1 Intuitive Pictorial Representations of Text

The weather map was already given as an example of a pictorial representation. Charts, diagrams, graphs, tables, guides, and maps comprise an enormous accumulation of material that all involve pictorial representations of information. This group of material was described by Philip Morrison as "cognitive art." These types of pictorial representation usually use some data that is in numerical form. However, it would be possible to describe the numerical data in textual form. For example, a weather announcer on television verbally gives the weather forecast.

There are other examples of pictorial representations of non–numerical data. In software design, flow–charts and Nassi–Schneiderman charts [Senn 84] are pictorial representations of algorithms. Military plans use a large set of map symbols to represent different objects. These plans show location, timing of movement, and expected goals. Again, these pictorial representations can be described textually. The software descriptions can be written in pseudo–code, and the military plan can be described textually as is frequently the case in newspaper articles or intelligence reports about the battle.

To go one step further, there are numerous examples of how pictorial representations can be used in the place of *full* natural language text. One example is sign language for the deaf [Miles 88]. Another is children's books in which the pictures tell more of the story than the simple text.

Although there are many examples of pictorial representations, can a picture be generated from natural language text? Could one manually accomplish such a task? It may not be immediately obvious exactly how to generate a picture to *represent* the meaning of some text, but most of us could after a little time thinking about it. A popular game, *Pictionary*, is based exactly upon this concept.

The next step in thinking about this process would be: if it were possible to generate pictures that generally captured the meaning represented in the text, could a particular picture be discerned from among other pictures that had subtle, yet distinct, differences in meanings (assuming some limited initial training by viewers on a *pictorial vocabulary*). If such a picture could be generated, could the translation process be formally described?

1.2 A Formal Study of the Relationship between Text and Pictures

Although, intuitively there seems to be some relationship between natural language text and a picture representing that text, a formal description of the relationship is lacking. A specific description of such a process to convert "text to pictures" is needed.

This thesis looks at a restricted problem of the inter–relationship *from* text *to* pictures—the problem of generating a pictorial representation from natural language text. The thesis does not attempt to handle natural language generation from pictures. There are several reasons for this, which are described fully in section 3.4, "Restrictions and Limits of this Research." This is primarily because the process of converting pictures to natural language text relied upon areas of research that are still being developed (i.e. natural language generation and visual representation).

Several types of natural language expressions are examined in this thesis, such as: nouns, relative clauses, conjunction, number, some verb features, spatial expressions, and temporal expressions. However, the *mapping process* is not uniformly complex among the different natural language components. This should not be surprising. It should not be assumed that pictorial representation theory should be any less complex that natural language theory.

The investigation leads to descriptions of how to develop a pictorial representation for each type of natural language expression investigated. Object nouns, such as 'cat' or a 'table,' can be represented with an icon, given a little thought. But, how are other natural language components, such as prepositions represented? And, what type of knowledge is required to generate a pictorial representation?

(1) There is a cat on the table.

In sentence 1, icons can pictorially represent the nouns 'cat,' and 'table.' However, there does not appear to be a universally understandable icon that would represent the preposition 'on'? A more useful method would be the placement of the two icons to adequately represent the concept *on*. A simple example, may look like figure 1.1. The cat icon is located in position (3,3), and the table icon in position (3,2). The 'on' relationship

is shown by the location and proximity of the cat and table icons.

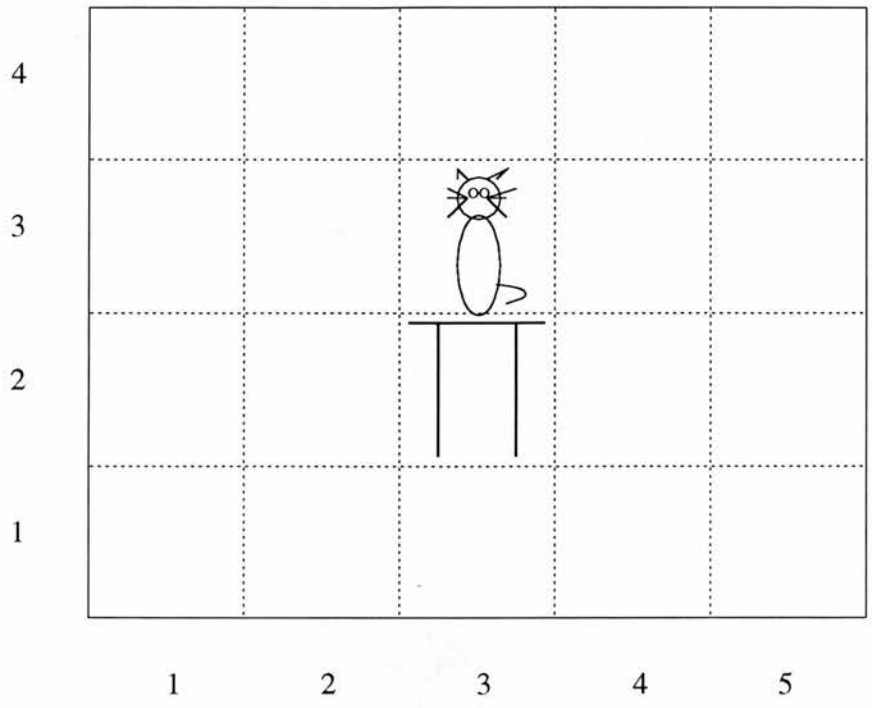


Figure 1.1: Example of Pictorial Representation of a Preposition

From the brief example given in the previous paragraph, one can see that a pictorial representation for prepositions must be handled differently than it is handled for nouns. Temporal expressions, conjunction, quantification, scoping, relative clauses and verb features also require different techniques to represent them than how nouns are handled. These are precisely the type of pictorial representation problems to be described in this thesis. Also for each type of linguistic expression, the type of knowledge that is required for the text to pictures translation process will be described. In my working system, the domain–independent knowledge is incorporated into the system, while the identified domain–dependent knowledge is isolated to user–defined modules.

Finally, I approached this investigation with the belief that it is better to categorize what types of natural language expressions can be translated to a pictorial representation and how to facilitate the translation process, rather than to say that the *entire* concept of converting natural language to a pictorial representation is flawed, because a few types of expressions are difficult to represent.

1.3 The Need for a System to Convert Text to Pictures

Very little work has been done that identifies how text can be represented pictorially. This is probably because much of other previous work describes possible techniques, but did not actually use or build a working system. A working system would allow text-to-pictures relationships to be found by experimentation.

The last section stated the restriction of converting from text to pictures and not vice versa. This restriction is important because it allows a working system to be built. I found that several concepts were discovered only after building and experimenting with a *working system*. It is only by the use of the system and using the feedback to improve the system, that one can really start to identify and understand these text–picture relationships.

The need for a working system for experimentation is clear. However, there are also a number of benefits of a text to pictures system. One feature of a text to pictures system is to present information in way which facilitates rapid interpretation and integration of data by humans. Rapid interpretation of large amounts of textual data is difficult. Integration of textual data is a slow process and may even be impossible with large amounts of textual data. Pictorial representations seem to allow for quicker interpretation and better integration [Tufte 90].

There are also a number of possible applications of a working system. One application that is implemented and described in this thesis is the disambiguation of sentences. Given a truly ambiguous sentence, the system will attempt to draw a picture of each possible meaning of the sentence. This could be used for text processing systems that interact with users who do not have a linguistic background. The user would simply have to point the mouse at the *picture* representing the intended meaning. This is certainly easier for a non–linguist user to decipher than a parse tree or logical form.

Two other possible applications are described in this thesis, although they are not implemented. One is a system operating with multiple natural languages by using pictures as an inter–lingua. The other described application is extending the system to work with large corpora of textual messages and using a data fusion technique to graphically display events that are contained within the messages. Many professionals such as medical

doctors, lawyers, financial analysts, analysts of police messages, or any research group that must review large amounts of textual data could benefit from a system that can display large amounts of a data in a way that is easier to discern information than from text directly.

1.4 Thesis Issues to be Addressed

There are several questions that are important to the issue of the inter–relationship of text and pictures. The main questions to be addressed by my thesis are:

1) **Can a pictorial representation be developed to adequately represent text?**

This is the main issue to be addressed by the thesis. The pictorial representation concept is presented in chapter three. A demonstration of this pictorial representation concept is shown for varied natural language expressions throughout the thesis. The pictorial representation used was tested by several people, whose recommendations were implemented, to make the representation an adequate and useful one.

2) **Can the inter–relationship between natural language text and pictorial representation be described?**

Given the pictorial representation proposed in this thesis, the translation process from text to this representation must be described formally. This issue is expanded upon in questions 3 and 4.

3) **If the inter–relationship can be described formally, can a working system be built using this description?**

An actual system must be shown to use the translation process from text to a pictorial representation. The system design is described in chapter four of the thesis.

4) **In detail, *how* are specific types of natural language expressions converted to a pictorial representation?**

The thesis investigates the capability of the pictorial representation by applying it to varied types of linguistic expressions. The translation process for each different (linguistic) type of expression is described. Chapters five through eight address the conversion

processes in detail.

5) **What knowledge (domain-dependent or domain-independent) is required for the conversion process of text to pictures?**

When designing a system, transportability is an issue. Some domain–dependent knowledge is required for the conversion process. The system design has attempted to isolate domain dependent–knowledge modules from domain–independent knowledge modules. The domain–dependent knowledge that is needed is discussed in chapter 4, "The System Design," where the domain-dependent modules of the text–to–pictures system are fully described.

1.5 Description of Following Chapters

Chapter two reviews previous work that correlates text with pictorial representation. Text processing and representation schemes are briefly looked as they are relevant to the overall text to pictures process. Specific work in text–pictures systems, such as Mel'chuk's work with the RITA system, Pineda's work with GRAFLOG, and other natural language systems that use graphics are described. It concludes by listing the areas of further research that are needed. This thesis attempts to start research into those areas.

Chapter three contains the central thesis idea. Specifically, it describes the pictorial representation concept: the fundamental idea, the process, examples, and a formal description. It defines a terminology of pictorial representation. The restrictions, limits and advantages of this concept are presented.

Chapter four describes the design of the text to pictures system. The pictorial generation process is specifically described. The components include: the language engine, a constraint builder, CSP (Constraint Satisfaction Problem) solver, a graphics module, knowledge sources for language processing, sentence constraint module, other knowledge sources that are required for the pictorial generation process, and the icon library. This chapter shows a data flow diagram, and describes the process of each of the components. This chapter also describes evaluation of how good the pictures were at representing the text. The chapter concludes with a description of the user feedback and how the pictorial representation evolved because of that feedback.

Chapters five through seven describe in detail the pictorial representations of several types of natural language expressions. Chapter five describes several general expressions to include noun phrases, the concept of object representation, a method of representing conjunction and quantification pictorially, and some verb features. Chapter six describes the use of spatial expressions in pictorial representation. Chapter seven looks at temporal expressions.

Chapter eight discusses the general problems of ambiguity and vagueness in both text and pictures. These problems include partial information (which still may be able to be displayed), insufficient information, and truly ambiguous information. An application of the Text-To-Pictures System, to display several pictures that represent each of the possible meanings of an ambiguous sentence, is shown.

Chapter nine looks at two possible applications of the pictorial representation concept. One application is a data fusion technique to pictorially represent data from a large number of natural language texts. Discourse, data abstraction, and new pictorial representations are discussed. The other section describes how the system might work with multiple natural languages and the advantages of using pictorial representation as an inter–lingua.

Chapter ten summarizes the thesis and re–addresses the main questions. A discussion of the main contributions of the thesis is given, and areas of further research are suggested.

Chapter 2

Review of Previous Work

The three major applicable areas of previous work to this thesis are *text–pictures systems*, *text processing*, and *representation schemes*.

Two projects concerning *text–pictures* relationships are summarized. Igor Mel'chuk, Academy of Sciences USSR, led the early work in a system called RITA that demonstrated relationships between some simple geometric shapes and Russian text. Luis Pineda developed a system called GRAFLOG that can be used in computer aided design that makes use of the relationships between text and pictures. Both systems are summarized.

Also discussed is the use of graphics as an aid in natural language systems. Graphics have been used in this area in an attempt to disambiguate natural language. Many of the graphics are simplistic and are not true pictorial representations of the text, rather the graphics is used as an aid to accompany the text.

Text processing is of some interest because that area uses the concept of converting the text to a representation scheme and then a further process acts upon the representation scheme. In this thesis, that concept is used, with pictorial generation being the further process.

Although, there are many text processing systems today, the thesis looks at a smaller set of text processing systems that were *available* for the author to use from the beginning of his research. Naomi Sager and her students led the initial research in text processing. Sager did early work concerning processing of medical text. More recently, two of her students have been responsible for building large scale text processing systems for the

US Defense Department funded by DARPA. Ralph Grishman of New York University is the principal author of the PROTEUS system. Lynette Hirschman, Martha Palmer, and Deborah Dahl developed the PUNDIT system—a system which summarizes Navy casualty reports. The Core Language Engine text processing system developed by SRI/Cambridge is also described. This is the text processor that was chosen for my thesis project. Other text processing systems are briefly mentioned.

The reasons for using an *intermediate representation scheme*, from the text to the pictorial representation, are described. A brief look at current *meaning representation schemes* is given. This is followed with a specific look at the Logical Form representation scheme used in the Core Language Engine.

I conclude this chapter by identifying where the research work in text–to–pictures systems should be extended, which is intended to be partially accomplished by this thesis.

2.1 Text–Picture Systems

2.1.1 Mel'chuk's RITA System

The relationship between text and pictures has been recognized for some time. One of the earliest experimental efforts was the RITA system (pictorial Representation - Information - Text - Author) [Mel'chuk *et al* 75].

Mel'chuk's system consisted of a language processor, a semantic graph meaning representation, and a composition processor. The system could convert text to pictures, as well as, pictures to text. The graphical input consisted of a simple picture of one, two, or three circles on a screen–the size of the circles and the location of the circles could vary. The textual input was Russian sentences of less than 45 words using a set of descriptors and comparisons to describe the "world." This experiment demonstrated a system that captured a simple relationship of spatial prepositions between text and pictures.

(2) A larger circle is located in the top right corner, and a smaller circle in the bottom left one.

Sentence 2 is processed by the RITA language processor, which involves a morphological analysis, a surface syntactic analysis, a deep syntactic analysis, and a transition to

SemP (which is a semantic network representation scheme). The SemP is decomposed into predicates (complex predicates are further decomposed into elementary ones). These predicates are used with a set of relations concerning the domain to form the "characteristics of the picture." Figure 2.1 shows an example taken from [Mel'chuk *et al* 75]. The example shows both the "composition" (the picture) and the "SemP" (the intermediate representation scheme from the text to the pictures).

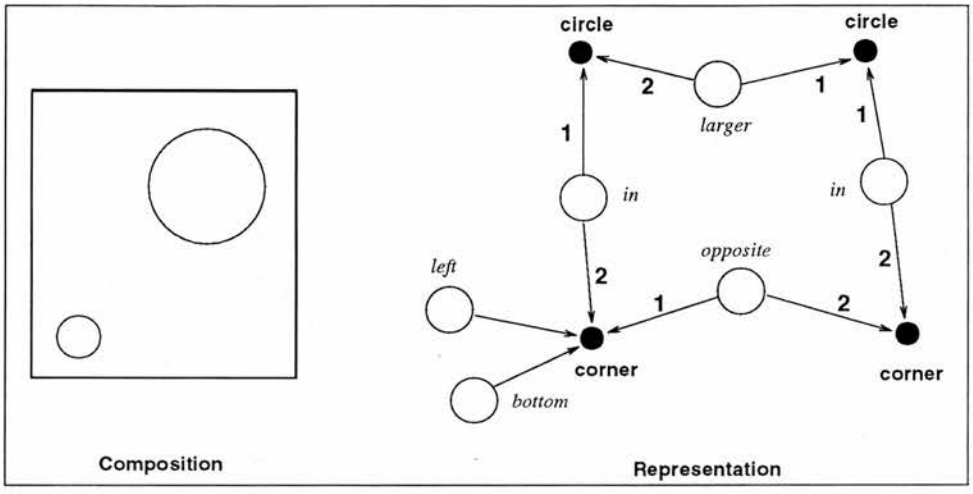


Figure 2.1: RITA Representation and Output

One difficulty is that the set of relations could not be easily modified to work with domains other than the three circles. The set of relations was basically a table of information that listed information concerning each circle, such as "radii ratio between 2nd and 3rd circles," or "the first circle intersects the 2nd circle." Each of these statements could be either true or false, or would have have a numerical value. This clearly attempted to establish a relationship between text and pictures, however, a generalized approach to handle a wide range of relations would be an improvement.

This work provides two basic ideas to build upon: first, using an intermediate representation scheme between text and pictures; second, the concept of a set of elementary concepts, or using complex concepts and translating them into elementary concepts.

2.1.2 Pineda's GRAFLOG System

More recent work has surrounded the relationship of natural language with computer aided design. Text could be used as commands, as questions, or even graphically represented. Luis Pineda, University of Edinburgh, presented an experimental interactive graphics interface, GRAFLOG, in which drawings receive linguistic interpretations [Pineda *et al* 88], [Cortes 89]. Again, this system could convert text to pictures, as well as, return a textual description of a simple picture. The system uses symbols to represent the "objects" and the locations of objects represent the relationship between the objects. GRAFLOG uses the data base structure within Prolog to represent the objects and relations. The actual graphical software uses Prolog calls that are binding with GKS[1] calls.

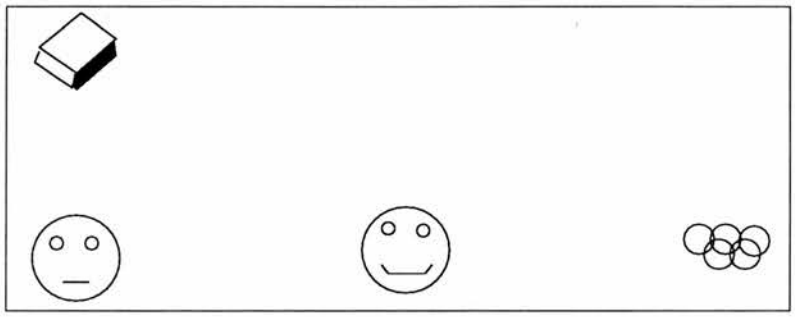


Figure 2.2: GRAFLOG Representation and Output

(3) If a game is to the right of Luis then, he wins it.

(4) If a book is above John then, he reads it.

These sentences (3) and (4) are represented in figure 2.2. GRAFLOG provides a powerful formalism using logic to describe and interpret pictorial representations. This could work very well in a CAD environment. However, using *one* picture to represent *multiple* (sometimes conflicting) relations is difficult, and in some cases, impossible.

GRAFLOG makes uses of a set of spatial relationships to represent other relationships. A problem arises in the example shown in figure 2.2, taken from [Cortes 89]. Luis is the

[1] GKS or the Graphics Kernel System is the prominent graphics protocol standard [Hopgood 83].

happy face which is located in the center of the picture, and John is the face on the left of the picture. The games are to the right of Luis, so he wins it. The games are also to the right of John. Did John also win the games? It is possible that once one starts using a method of interpretation for a relationship, that the same method of interpretation is used for the remaining relationships shown in that picture [Klein 90].

In addition to the possible high degree of false implicatures arising from the GRAFLOG representation, GRAFLOG lacks a set of restrictions to represent text in ways that is readily understandable. This is key to identifying and exploiting inter–relationships between text and pictures. Different facets contained in natural language should be captured in the pictorial representation.

Pineda's basic method of representing relations by placement is a very good idea, and is similarly used within my system. A set of constraints restricting the placement, so that it is not completely open-ended could facilitate understanding and hopefully reduce false implicatures.

2.1.3 Other Systems that combine Graphics with NL Processing

There are other recent systems that work with spatial relationships and produce a picture. A group in Kyoto, [Nishida *et al* 88], [Yamada *et al* 88], [Yamada *et al* 92], built the SPRINT system that generates 3-D images of text. The system uses the spatial prepositions, a set of heuristics concerning geometric properties, and simple real–world knowledge of objects (e.g. location, size, orientation, shape, et cetera). Notably, this system has a set of heuristics that make use of real-world knowledge, which can modify a meaning of a spatial expression. This work is restricted to a small set of spatial prepositions.

Graphics could also be used with text generation, as is the case in the WIP project [Wahlster *et al* 91a],[Wahlster *et al* 91b], [Wazinski 92]. WIP is a knowledge-based text and pictures presentation system, and is demonstrated with an example of generating instructions for use of an espresso machine. A planning module takes into account a given explanation graphic, the viewpoint, and what parts of the espresso machine are in view, and determines an *appropriate* explanation making use of the explanation graphic.

Marks and Reiter [Marks & Reiter 90] describe their system FN/ANDD and the problem

of false implicatures in text and graphics. They apply Grice's maxims to reduce unwanted implicatures in a generation of electronic circuit schematic diagrams.

Systems, such as the ones developed by [Feiner & McKeown 90], [Moore & Swartout 90], use graphics for explanation, pointing, focusing, or supplementing text processing systems with additional information. The COMET system (COordinated Multi-media Explanation Testbed) uses graphics for explanation purposes in a system that generates directions for equipment maintenance and repair. Moore and Swartout use a graphical pointing technique so the user can guide the scope of explanations. These two systems are not generation of pictures from text. However, they do show that use of text with pictures has many advantages.

The graphics of these systems are quite different from those in this thesis. The pointing system by Moore and Swartout is really highlighting regions of text. The COMET system works within an test equipment domain, and shows drawings of test equipment in different positions. Marks and Reiter's system works with schematic diagrams. All of these systems work within specific domains and can take advantage of knowledge and graphics unique to their domain. The graphics used within this thesis are icons that attempt to represent text in a much more broad domain.

2.1.4 Relevant Work on Graphical Presentation

Other work has been done concerning graphical techniques of representing information, such as Charnoff faces [Wainer & Thissen 81] and aircraft cockpit displays. However, these works do not specify any relationships with text, rather they concentrate on graphical relationships with numerical data.

Tuft's book, "Envisioning Information" [Tufte 90], is a study which attempts to list the ingredients for a good display of information. Concepts of representing multi–variate data and how to *pack* data are presented. Packing techniques include: micro/macro readings, layering and separation, small multiples, color and information, and narratives of space and time. Some of Tufte's work motivated my pictorial representation in the search of *best* ways to represent information pictorially. These are evident in use of separation, and narratives of time, which will be shown in Chapter 6, "Spatial Expressions," and Chapter 7, "Temporal Expressions."

2.2 Text Processing

2.2.1 What is Text processing?

Text processing is the automated conversion of text to a representation scheme that is useful for some type of further processing [Spark-Jones & Wilks 85].

Figure 2.3 shows a simplified schematic of "natural language processing" in information management [Sager *et al* 87].

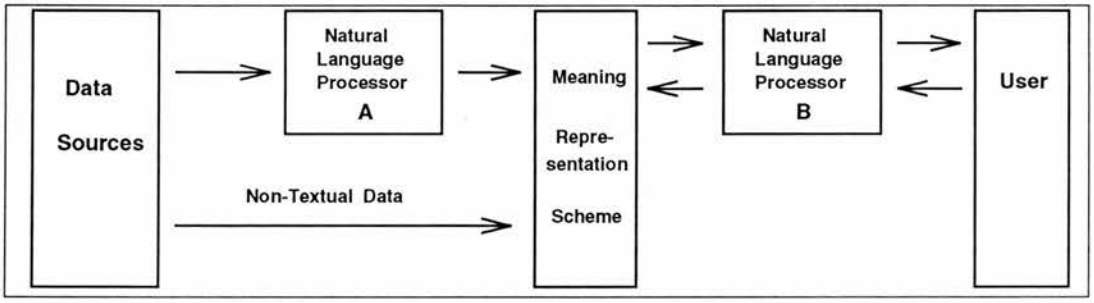


Figure 2.3: Natural Language Processors

Five years ago, much of the natural language processing research has focussed on developing a "natural language front-end" to allow a user to use natural language to query a pre–built data base consisting of tabular data. This type of system is commonly referred to as a **Natural Language Interface**. This is depicted as "natural language processor B" in figure 2.3. Commercial examples of such systems are Q AND A by *Symantec* and CLOUT by *MicroRim*. These systems are direct descendants of the more academic work in the LIFER and LADDER programs [Hendrix *et al* 78], [Morris & Sagolowiz 77], [Sacerdoti 77], [Sagalowicz 77], [Hendrix 77]. Although these systems are *natural language processors*, Sager distinguishes these systems from **text processors**, which are shown in 2.3 as "natural language processor A." Text processors handle primarily declarative text, where the natural language interface primarily handles interrogatives. Although, it is true that text processors may be able to handle interrogatives, and a natural language interface to handle declarative text, it was found that large scale natural language systems tend to handle declarative or interrogative text much better than the other types of text because they are built primarily to handle one type of text [Montgomery & Neal 91]. This project requires a natural language processor that converts text (primarily declarative text) to

a representation, rather than interfacing (in natural language) via a set of queries with some data base. Therefore, a natural language system that is a text processor is needed.

Information is in many forms, however, a large amount of it is in textual form. "If the amount of (textual) information to be translated was growing rapidly, the amount (of text) to be stored and retrieved was growing even faster" [Winograd 83]. If a text processor could be built to convert natural language text of varied subject matter into an intermediate representation scheme, which was suitable for retrieval by a further process such as a natural language interface or a graphics module, this would be a major step towards achieving a system to rapidly and effectively present information. Such a system is needed for this thesis project and several possibilities will be identified.

Given a definition of text processing, the work of Sager, Grishman and Hirschman, and SRI Cambridge is presented.

2.2.2 Linguistic String Project

The Linguistic String Project was developed by Naomi Sager at New York University. This was one of the first large text processing systems. Sager calls her approach "natural language information formatting." She describes the problem as "how to structure free running text so that its content is made accessible for processing" [Sager *et al* 87]. Basically, the system breaks apart the sentence and stores the appropriate portion of the sentence into slots associated with meaning [Sager 78]. The slots are in a table format or structured data base. Her system was designed to use narrative medical text.

Although this system works well within in its domain, it does not work well in handling text of new subject domains. This is primarily because of its domain–dependent representation scheme. I need a text processor that represents the text in a general framework.

Other work based upon Sager's approach includes the author's MSc thesis which describes processing of narrative English text containing varied subject matter [Ludlow 88]. The system formats the text into meaning slots as did Sager's system. However, the author's system uses domain–independent slots to handle varied subject matter. The actual filling of the slots is accomplished by a case–mapper module. The mapping process for each slot varies. Some of the mappings are based solely upon grammar, some require lexical

knowledge, and some of the slots require semantic knowledge to be filled. It is important to note that the system maximizes the use of syntactic knowledge and only uses semantic means to determine the mapping of a slot as a last resort. However, this system was not chosen as the natural language engine for this thesis, as it had a limited linguistic coverage. I wanted to concentrate most of the time on developing the pictorial representations of text, rather than expanding a natural language system.

2.2.3 PROTEUS and PUNDIT

Grishman's work with PROTEUS and Hirschman's with PUNDIT are two examples of large scale text processing systems [Grishman & Hirschman 86]. Both projects have been funded by DARPA and were developed to "understand" Navy Casualty Reports.

The PROTEUS System (PROtotype TExt Understand System) is an extension of Sager's Linguistic String Project. The system uses context-Free BNF definitions plus restrictions, that allow context sensitivity [Grishman 87].

The PUNDIT system (Prolog UNDerstand of Integrated Text) is also related to Sager's Linguistic String Project and Grishman's PROTEUS system [Palmer *et al* 88]. However, PUNDIT allows for a powerful framework for writing definite clause grammars [Pereira & Warren 80]. The system also incorporates a discourse module over an entire message to resolve anaphora. This discourse module uses linguistic cues, contextual information and domain information such as dependency, association relationships such as part/whole(a sac..the pump), property(oil..the pressure), possession(a ship..the crew) and isa–hierarchy(sac..unit).

These systems were more mature than the Linguistic String Project. However, the main drawback was that the representation or output from the text processors were domain–dependent slots that were filled with the appropriate text, rather than a generalized representation scheme.

2.2.4 Core Language Engine

The Core Language Engine (CLE) was developed by SRI/Cambridge. This system was funded by a large consortium and the project lasted three years. The focus of the CLE

project was to address linguistic and computational research issues involved in the development of a language engine–a general purpose system for deriving representations of the meanings of English sentences [Alshawi *et al* 88].

The CLE is a modular system using the concept of processing the text in several stages. These stages consist of a morphology stage, a parsing stage, a semantic interpretation stage, sortal filtering, quantifier scoping, reference, and a final disambiguation stage. These stages are in order of the processing. After the semantic interpretation stage, the representation is somewhat like a logical form. It is called an **unsorted unscoped quasi logical form**. Each stage provides more information or reduces the number of possible interpretations of the sentence. "The final result is a set of fully specified **logical forms** that represent the possible literal meanings of the input sentence" [Alshawi 90]. This is the normal case. Sometimes these modules or stages cannot resolve what they intend to do (i.e. unresolved reference for the pronoun, the scoping rule does not work). In these cases, the system passes the quasi–logical form on to the next stage. Therefore, the final output of the CLE could include quasi–logical forms as well as logical forms.

The linguistic coverage of the system includes (for English) [Alshawi *et al* 88]:

- Several major-clause types: declaratives, imperatives, wh-questions, yes–no questions, relatives, passives, clefts, there clauses.
- Verb phrases: complement subcategorization, control verbs, verb–particles, auxiliaries, tense operators, some adverbials.
- Noun phrases: pre–nominal and post–nominal modifiers, lexical and phrasal quantifiers.
- Coordination: conjunctions and disjunctions for a wide range of noun phrases, verb phrases and clauses.
- Morphology: inflectional morphology, simple productive cases of derivational morphology.

The CLE uses a GPSG (Generalized Phrase Structure Grammar) approach and is based upon concepts presented in [Gazdar *et al* 85], [Pereira & Shieber 87].

The linguistic coverage of CLE is very good, and provides a logical form representation that is generalized and can handle text of a wide variety of subject matter[2]. Also, the logical form representation is one that is in "the common domain," so others would be familiar with the representation. Finally, the CLE is easy to use, particularly in customizing the lexicon, modifying the grammar, or adding sortal restrictions. For these reasons, I chose CLE as the text processor for this project.

The CLE project has been extended to improve coverage, performance, and to handle discourse. That project is called CLARE (Core Language And Resolving Engine). The system I used for this research, was actually the CLARE version of the project. For the purposes of this thesis, the names CLE and CLARE are interchangeable.

2.2.5 Other Work in Text Processing

Although there are currently several text processing systems, I had to choose a text processing system to use at the beginning of my research. Therefore, the previous mentioned systems were looked at. Other work in Text Processing has been summarized in a Survey of Text Processing Systems [Onyshkevych 88]. Finally, many recent systems were demonstrated at the Message Understanding Conferences (MUC) held by DARPA [Sundheim 91]. At the MUC-3 conference (1991), twelve systems were evaluated. At the MUC-4 conference (1992), they expect to have over twenty systems to evaluate.

Most of the MUC systems are knowledge driven systems, which make heavy use of domain–dependent methods to represent the text. Also, the evaluation of MUC systems was to store the data into domain–dependent slots. For example, in the two domains used (equipment casualty reports and terrorist reports) the systems were to store appropriate pieces of text into slots such as "benefactor of terrorist action," "terrorist organization," or "engine part." This is very similar to slot representation of the text as Sager did with medical language processing. Again, this would rule out most systems in MUC to be useful as the text processor needed for this project.

[2] The CLE representation scheme is described in the following section 2.3.3, "CLE Logical Form Representation Scheme."

2.3 Meaning Representation Schemes

The central idea of this thesis is to investigate, identify, and exploit the interrelationships between text and pictorial representations. Given the previous work in text processing that converts the text into some representation scheme, this thesis proposes using a text processing system to convert the text to a representation scheme which is an intermediate step in the overall conversion of text to pictures. A pictorial representation generator converts the intermediate representation of the text into the final pictorial representation.

Figure 2.4 shows the overall *Text–To–Pictures System* (TTPS). Notice the similarity to figure 2.3, which shows two types of natural language processors. The natural language interface to the data base is replaced with a pictorial representation generator.

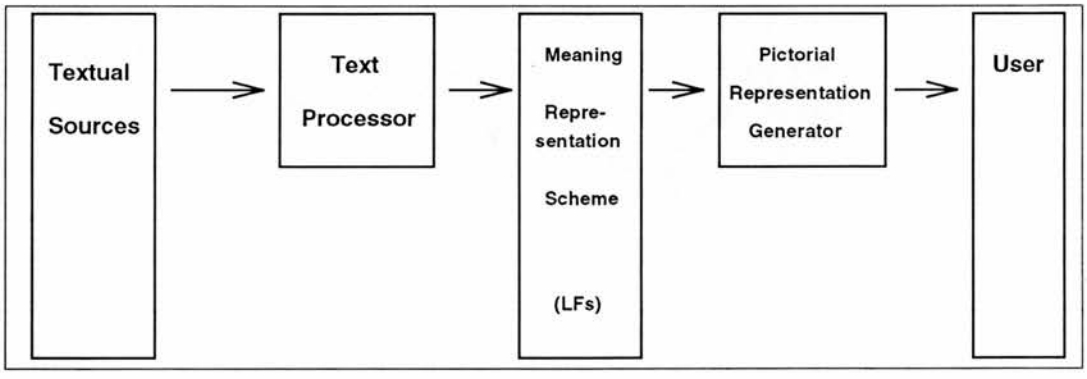


Figure 2.4: Text to Pictures System

My text–to–pictures system makes use of a intermediate representation scheme, as did the RITA and GRAFLOG systems. But, is an intermediate representation scheme needed? Or could text be directly represented by pictures?

I believe an intermediate representation scheme is needed because it allows:

- (1) a modular design to be used;
- (2) existing text processing research to be used; and
- (3) the pictorial generation process to use any text processing system that generates the same representation (in this case, logical forms).

It is possible that a specialized system could be built to directly convert text to pictures. However, such a design would intertwine all of the processes. It would be extremely difficult to isolate different types of knowledge required and associate them with a specific process, unless a modular design is used [Grosz 83], [Martin *et al* 83]. Updates, changes, and modifications to the system would be unwieldy without a modular design. To achieve a robust system, capable of converting text of a wide range of subject matter to a pictorial representation, a modular design is needed.

The second reason for using an intermediate representation scheme is to make use of existing research work. All of the text processors previously described took large teams several years to develop. A text–to–pictures system should capitalize on this existing software. The only way to use any of these text processors is to use their "output" as the intermediate representation scheme in the overall text–to–pictures process.

Finally, my pictorial generation system and technique are more powerful, if the pictorial generation process can work with any text processing system that converts text to a logical form representation. To do this, the pictorial generation process must be isolated from the text processor.

The pictorial generation process described in this thesis uses a well known existing text processing system, the Core Language Engine. In theory, the text processor could be changed, and the overall text–to–pictures process would be unchanged. An intermediate representation scheme is needed to be the "interface" between the text processor and the pictorial generation processor.

There are several reasons for using an intermediate representation scheme. The next section takes a closer look at the requirements for an intermediate representation scheme, and some current schemes.

2.3.1 Intermediate Representation Schemes

The Intermediate Representation Scheme is a critical component to the overall system. Any representation scheme for representing text should attempt to meet the following requirements exemplified by W. A. Woods' Meaning Representation Language [Woods 73], [Woods 78]:

a) it must be capable of representing precisely, formally, and unambiguously the human reader's interpretation of a sentence;

b) it should facilitate an algorithmic translation of the text into the representation; and

c) it should facilitate subsequent intelligent processing of the resulting interpretation or representation.

However, these requirements are not enough if one wishes to use the representation in applications. I propose two additional requirements.

The first requirement is handling text of varied subject matter. It is simpler to have a system that operates within a limited subject domain, but this is an unrealistic constraint for a system that would be of any value to many professionals(students, analysts, et cetera) that do involve processing text of varied subject matter. One cannot restrict a priori the information that might be processed.

Coupled with the varied subject matter requirement, the system must also be able to handle new information. Often a new concept is of more interest than the often uninteresting information that fits some pattern. This prohibits the use of a scheme that makes heavy use of domain–dependent knowledge, unless it is absolutely required.

The system's requirement is an intermediate representation scheme that works both in representing the text and that can supply the graphics processor with useful information. Most of the representation schemes were only designed with the emphasis of representing the text and not being used for a further process. The pictorial generation process will serve as a good test of the viability of the intermediate representation scheme to satisfy Woods' third requirement.

2.3.2 Current Meaning Representation Schemes

"In computational linguistics, our objective is normally not just the parsing of a sentence, but figuring out what the sentence means" [Grishman 86]. What a sentence means is precisely what should be stored in the representation scheme. There are numerous methods of representing the meaning of a sentence such as frames [Minsky 81], conceptual dependency (scripts, goals, plans, themes) [Schank 80], various meaning representational

languages [Woods 78], and logical forms [McCord 82], [McCord 87], [Pereira 83].

The problem is that some of these representation schemes either can not represent text of varied subject matter without the representation scheme being redesigned, or that the representation is unstructured and does not adequately facilitate subsequent intelligent processing (i.e. the pictorial representation process). Briefly, a representation scheme such as *scripts* in conceptual dependency works only within a limited subject domain. Roger Schank describes a restaurant script that contains a sequence of commonly expected events. If one goes to an opera, the restaurant script would be of little value. A new script must be built. Scripts can adapt to other subject domains but concentrate on things that can be grouped together as repetitive. Although scripts build upon a set of primitives, they do not work well with events that only occur once or have never been seen before. In fact, there are no means to adequately represent an event until one has a script to represent it. Another representation scheme was Mel'chuk's RITA system, which used *semantic nets*. This adequately captured the relations between objects. Other schemes, such as *logical forms* can represent text of varied subject matter. Pereira's CHAT-80 system used this representation. The advantage of such a representation is that it can be used in applications which involve inferences. Inferencing may be useful in *meaning retrieval*, but the logical form must be rich enough to enable direct retrieval of meaning (e.g. spatial, temporal information).

There are many representation schemes, and some of them can adequately serve as an intermediate representation for the text to picture process. However, another requirement was to have access to an available text processing system that produced the desired intermediate representation scheme.

In summary, the requirements for the intermediate representation scheme were that it be suitable for both subsequent graphics processing and handling of varied subject matter, and that a text processing system that generates this representation scheme be available. Therefore, I chose the logical form representation [Alshawi 90] used in the Core Language Engine developed by SRI. A brief look at the CLE logical form representation scheme is given in the next section.

2.3.3 CLE Logical Form Representation Scheme

The Core Language Engine produces a logical form (LF) representation for a sentence. The LF is centered around the main verb. Actors and objects are arguments of the verb. The LF also includes scoping and shows attachment of modifiers and prepositional phrases. Using the requirements by Woods for a meaning representation language, an LF representation must satisfy the following:

- a) LF should be precise, formal, and unambiguous. Different natural language readings should yield different logical forms;
- b) LF should facilitate an algorithmic translation of the text into the representation. This to be handled by the Core Language Engine;
- c) LF should facilitate subsequent intelligent processing.

Many natural language systems only test the first two points. However, this system will *test* the knowledge representation scheme from all three perspectives–notably, goal "c" is evaluated by using this LF representation to generate pictures.

$\langle$ *lf_formula* $\rangle \rightarrow$ quant($\langle$*quantifier*$\rangle$, $\langle$*variable*$\rangle$, $\langle$*restriction*$\rangle$, $\langle$*body*$\rangle$)
$\langle$ *lf_formula* $\rangle \rightarrow$ quant($\langle$*wh_sense*$\rangle$, $\langle$*variable*$\rangle$, $\langle$*restriction*$\rangle$, $\langle$*body*$\rangle$)
$\langle$ *lf_formula* $\rangle \rightarrow$ [$\langle$*functor*$\rangle$, $\langle argument_1 \rangle$, ..., $\langle argument_n \rangle$]
$\langle$ *quantifier* $\rangle \rightarrow$ forall | exists | most | ...
$\langle$ *quantifier* $\rangle \rightarrow \langle$*variable*$\rangle \wedge \langle$*variable*$\rangle \wedge \langle$*lf_formula*$\rangle$
$\langle$ *wh_sense* $\rangle \rightarrow$ wh1 | count
$\langle$ *restriction* $\rangle \rightarrow \langle$ *lf_formula* $\rangle$
$\langle$ *body* $\rangle \rightarrow \langle$ *lf_formula* $\rangle$
$\langle$ *argument* $\rangle \rightarrow \langle$ *lf_formula* $\rangle$
$\langle$ *argument* $\rangle \rightarrow \langle$ *term* $\rangle$
$\langle$ *argument* $\rangle \rightarrow \langle$*variable*$\rangle \wedge \langle$*lf_formula*$\rangle$
$\langle$ *term* $\rangle \rightarrow \langle$ *variable* $\rangle$
$\langle$ *term* $\rangle \rightarrow \langle$ *constant* $\rangle$
$\langle$ *term* $\rangle \rightarrow$ kind($\langle$ *constant* $\rangle$, $\langle$ *restriction* $\rangle$)

Figure 2.5: BNF–like rules for the Logical Forms used in CLE and TTPS

I will give an example of the logical form representation, but first a formal definition of the logical forms is given. The syntax of the Core Language Engine Logical Forms

[Alshawi *et al* 88] is shown in figure 2.5.

(5) I saw the man on the hill with a telescope.

Figure 2.6 contains one of logical form representations for sentence 5. Using the syntax described in figure 2.5, a logical form is generated by the CLE that represents sentence 5.

```
[dcl,
 quant(exists,
       A,
       quant(exists,
             B,
             [and,
               [hill_HighGround,B],
               quant(exists,
                     C,
                     [telescope,C],
                     [with_Having,B,C])],
             [and,
               [man_MalePerson,A],
               [on_SpatiallyOnTopOf,A,B]]),
       quant(exists,
             D,
             [event,D],
             [past,[see_LookAt,D,speaker,A]]))]
```

Figure 2.6: Sample of Logical Form Representation

The logical form, given in figure 2.6, will be described.

The first token is 'dcl', which is means that sentence is declarative.

It is followed by a 'quant' group, which contains four components:

a) the actual quantifier;

b) the variable;

c) the restriction (which make the association to the variable);

d) the "body" (which is a recursive structure of other LFs).

One of the possible syntax descriptions for *lf_formula* is a *functor* and its associated *arguments*. 'On' is an example of a functor with two arguments. 'On' is translated here

as the relation:

```
[on_SpatiallyOnTopOf,A,B]
```

Which can be paraphrased as "A is spatially on top of B." The same is similarly true for the *with_Having* functor.

If one parses the structure further, it is seen that another functor 'see_LookAt' represents the *event D*. It is shown by the structure:

```
quant(exists,D,[event,D],[past,[see_LookAt,D,speaker,A]])
```

It is read as, there exists an event **D**, that has a relation *see_LookAt*, which relates the event **D**, the *speaker*, who is the actor of the event, and **A** (*man_MalePerson*) who is the patient of the event. There is also a temporal relation "past" which modifies the entire *see_LookAt* structure. In the CLE lexicon, some relations are two-place (intransitive verbs), some three-place (transitive verbs), and the 'give' functor is four-place (ditransitive verb).

Logical forms are not intuitively obvious to the unfamiliar, nor is it easy to read a logical form quickly. However, the representation gives a precise, and unambiguous representation of the text.

2.4 Areas for Improvements

What is needed for the text-pictures system is a conversion process from text to an intermediate representation scheme to a pictorial generation process, which will produce a pictorial representation. This system is described in the chapter 4, "The System Design."

The previous work described in this chapter can be combined to provide an excellent basis to build a text to pictures system. Specifically, the text processing and representation scheme of the Core Language Engine provide the foundation for this thesis project. This allows the focus of this report to concentrate upon the Pictorial Generation Process that uses the logical forms as input.

The previous text–pictures systems only made use of the object concept for nouns and some simple usage of spatial expressions. This thesis carries the work further by exploring

pictorial representations of temporal expressions, conjunction, relative clauses, number, scoping, negation, and some verb features. The next chapter describes the concept of pictorial representation for overall natural language text.

Chapter 3

Concept of Pictorial Representation

Can a textual sentence be represented pictorially? Does there exist some pictorial description that will represent to the viewer the intended meaning of the sentence? And more importantly, if a pictorial representation exists can it be formally described and does this pictorial representation meet the standard requirements of a meaning representation scheme?

This chapter attempts to answer these questions and to present the basis of a pictorial representation scheme. Some terms used in pictorial representation are defined, and the concept of the pictorial representation used in this thesis is given at a fundamental level. An example is presented, with a discussion of several issues concerning representation. The chapter concludes with the limits and restrictions, as well as the advantages, of the pictorial representation concept.

3.1 Terminology

It is important to have clear definitions of several terms that are often used concerning pictorial representation. For many of these terms, a precise definition is not found in any reference, and to make it more difficult, many people use these terms very loosely. Therefore, a definition of the relevant terms is introduced.

The **actual scene** in the world is what one would see with one's eye.

Images (or visual images) are not an actual scene–they are not the same as looking with the human eye. An image is a *representation* of an actual scene. They are an abstraction of what one sees. "Having an image of something is like seeing that thing by means of a pictorial representation of it" [Rollins 89].

Charts are a subset of visual images that display data. These include maps, diagrams, and graphs. These are not photographs, drawings, or scenes.

Visual Scenes are visual images of some event or state. It does not simply represent information or data. It represents a coherent event, state, or thought.

Mental Images are a representation that is cognitively relevant. This is a representation that the human uses. The pictorial representation in this thesis is not intended to produce mental images.

Vision is the process of how animals (primarily humans) see, which involves the mechanisms of the eye, the localization of process, and possible representations that the brain uses to understand the data that the eye provides [Marr 82], [Barlow 90].

Graphics is defined by [Meriam-Webster 86] as the the art or science of drawing an object on a two–dimensional surface. Therefore, computer graphics is the science of implementing drawings of an object via a two-dimensional set of pixels. This science includes topics such as three–dimensional graphics, hidden line removal, shading, anti–aliasing techniques, and animation [Watt 89], [Foley *et al* 90].

Pictorial Representation is a representation scheme that uses pictures, visual scenes, or mental images.

Icons are pictorial representations of objects [Meriam-Webster 86].

Given the definitions of pictorial terms, the next section defines the thesis concept of pictorial representation of text.

3.2 The Concept of Pictorial Representation

Given the previous terminology, *I am using a* **visual scene** *to pictorially represent text.* I am not using a mental image, and make no claim that this approach is cognitively

relevant. The representation attempts to meet the requirements of a "good" pictorial representation (i.e. capable to represent, minimize false implicatures, be understandble), which is defined in the following section 3.3.1, "Goal of a Good Pictorial Representation Scheme." The restrictions on what can be represented are given in section 3.4, "Limits and Restrictions of the Concept." First, the fundamental idea of the concept is presented.

3.2.1 The Fundamental Idea

The pictorial representation concept first assumes that *visual objects* exist. These are the nouns of the textual sentence[1]. They have an associated iconic representation whether the object is real or imagined. An argument against this assumption could be made, that some nouns cannot be represented by an icon. For now, this assumption will be used.

The representation uses several pictorial representation windows. A **pictorial representation window** (PRW) can display a set of visual *objects* and the *relationships* between these objects. This is a recursive definition, in that a PRW can contain a set of other PRWs. For example, the top level of the entire sentence would be a PRW that contains other PRWs and objects.

Given the concept of a Pictorial Representation Window and a set of visual objects represented by icons, the problem of pictorial representation is then reduced to:

(a) placement of the icons or other PRWs inside a containing PRW;
(b) modification of the icons;
(c) inclusion of new icons to symbolize relationships.

The problem of placement of the icons (and placement of other PRWs) inside a containing PRW is fundamental to the concept. Given an icon for a 'cat' and another icon for a 'table', and given that the "cat is on the table," the relationship among the two objects is *represented* by the placement of these icons. For example, the cat icon should be above and touching the 'table' icon. There are many other factors involved in producing a correct representation. They will be described in detail later. However, placement of the icons is fundamental to pictorial representation.

---

[1] I am not saying that an icon exists for every noun. I do believe that one exists for *most* nouns. I discuss this problem of visual object representation in section 5.1, "Object Concept."

(6) I see the cat on the table.

The modification of icons means the use of size, color, and emphasizing marks. In sentence 6, the representation needs to indicate the focus of the verb *see*. Therefore, the 'cat' icon would need to be modified to show emphasis. A pictorial example is given in the next section.

The inclusion of new icons to symbolize relationships is also needed. Many relationships can be represented by placement of the icons, or the modification to one or more icons. However, some relationships must be shown via a different means–by inclusion of an additional icon. This icon represents a relationship and not a visual object. In the previous sentence 6, a seeing event is given. This can be thought of as a relationship between the actor and the object. For example, this relationship could be displayed via icon representing seeing (e.g. an icon of a directional eye). One may come up with other icons, however the relationship must be represented by some additional icon that represents the relationship between the actor and the object.

3.2.2 An Example

An example of a pictorial representation generated by the Text-To-Pictures System is given for the following sentence:

(7) The man who drove the car saw the cat on the table.

Figure 3.1 is one of the pictorial representations for sentence 7.

Notice there are four "windows" that each represent a different piece of the total sentence. Some relationships are shown within a single window (such as "saw the cat") and other relationships are shown between windows (for example, the 'man' is the *same*[2] in pictorial representation windows PRW1 and PRW3. The main verb event is shown in PRW1. The associated time of the main verb event is represented in the attached pictorial representation window, PRW2. Each standard PRW showing an event or state has an associated time PRW. PRW3 represents the relative clause information, with PRW4 representing the time information, associated with PRW3.

[2] This is shown by using the same icon to represent the man. If the representation requires two men, then two different icons are used.

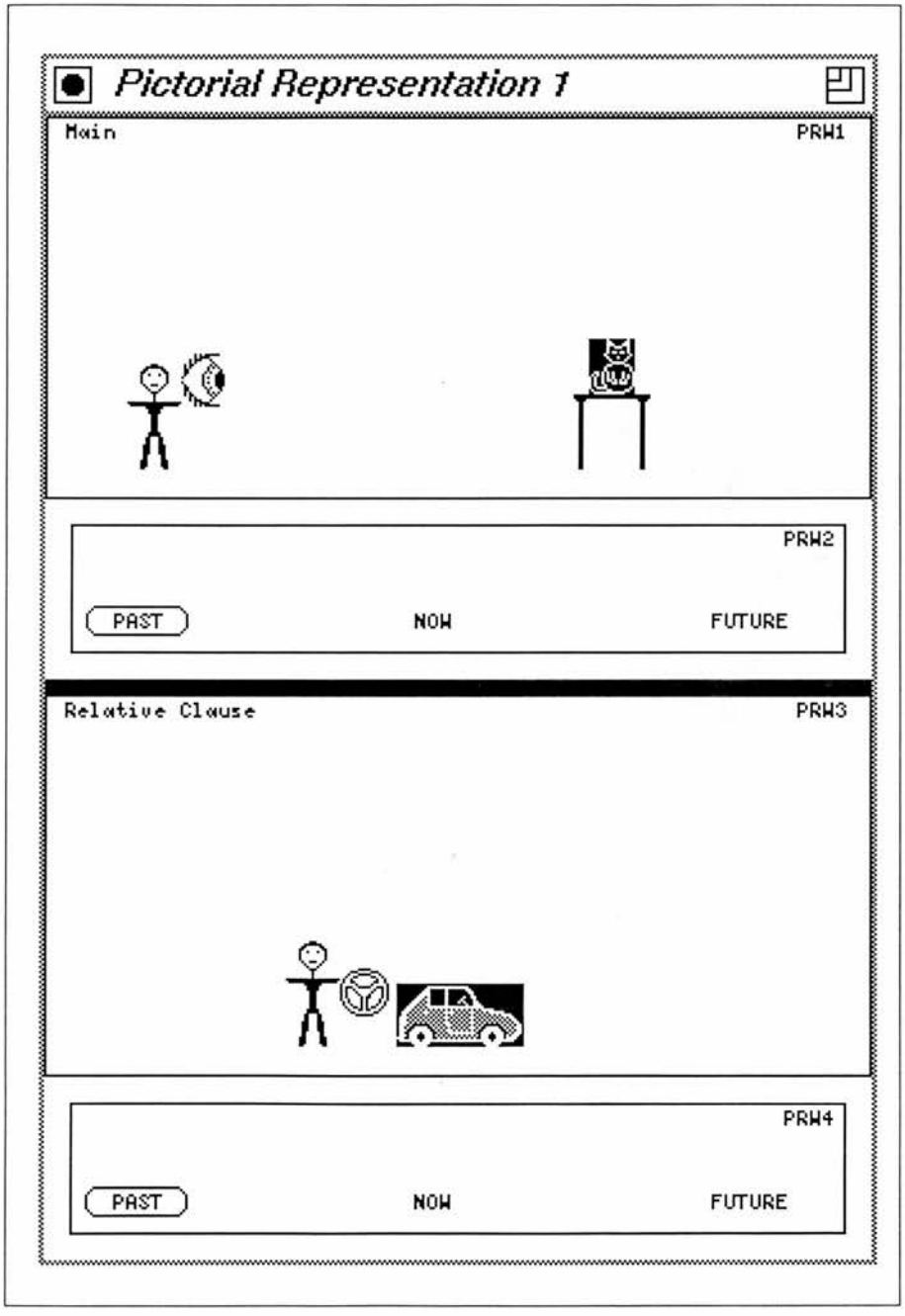


Figure 3.1: Pictorial Representation Example

Different windows are required because there is only one (implied) definition of representation that is defined within one window. This would prevent certain types of information from being displayed together within one window. For example, in the spatial representation windows, if something is 'left' or 'above' it is shown spatially as physically located left or above. However, in the temporal representation window, locations 'left' and 'right' are used to display temporal 'before' and 'after'. Therefore, the window is used to mark the boundary of a particular pictorial representation technique and subsequent interpretation. Much more on representation technique in Chapter 6, "Spatial Expressions" and Chapter 7, "Temporal Expressions."

There are several pieces of information that are displayed within this representation. For this example, the entire list of pictorial information is given in figure 3.2.

Before a formal description of the representation is presented, a brief look at the translation process of converting a textual sentence to a pictorial representation is given. A full description of the process is given in the next chapter, "The System Design."

3.2.3 The Process

To achieve a pictorial representation from text, the picture is generated by a multi-stage process. First, I assume a natural language processor has converted a sentence in a logical form representation. The next step is to look at the logical form representation and attempt to identify the visual objects and the relationships between the visual objects[3].

The process has two data structures at its disposal. First, the process makes use of an imaginary space that can contain an iconic representation of each object. This iconic representation on the imaginary space is ultimately displayed as a pictorial representation window as shown in the example. This imaginary pictorial space can contain unlimited number of icons. The second data structure is an icon and its size information. This can be used by the assumption that each visual object has an iconic representation[4].

Given the visual objects, a list of relationships, and the PRW and icon data structures,

[3] The details of the process of how to convert a logical form into a pictorial representation, how to identify visual objects and relationships, et cetera are given in Chapter 4, "The System Design." This section is to give the general concept of the conversion process.

[4] This assumption is an arguable point. See chapter 5, "General Expressions," for discussion that some objects do not have a visual representation.

```
1 Sentence:
     The man who drove the car saw the cat on the table.

5 PRWs:
     (1)  Man sees the cat on the table     [event]
     (2)  saw = past(see)                   [time]
     (3)  Man drove the car                 [event]
     (4)  drove = past(drive)               [time]
     (5)  Global PRW containing the other 4 PRWs

in PRW 1, visual objects are:
     (1)  Man
     (2)  Cat
     (3)  Table

and relationships in PRW 1 are:
     (1) see(Man,Cat)
     (2) on(Cat,Table)
     (3) *emphasis-as-object-of-verb*(Cat)

in PRW 2, time of event in PRW 1 is:
     (1) past

in PRW 3, visual objects are:
     (1)  Man  (same man as object 1 in PRW 1)
     (2)  Car

and relationships in PRW 3 are:
     (1) drive(Man,Car)

in PRW 4, time of event in PRW 1 is:
     (1) past
```

Figure 3.2: List of Pictorial Information

each relationship is to be described. The relationship generates a set of pictorial constraints on the overall picture. This is basically a translation process of converting a natural language relationships (e.g. from verbs and prepositions) into a set of pictorial constraints (e.g. above, near, bigger than, et cetera). This translation process is conducted by a Constraint Builder. The output is a list of "visual constraints" that act upon the icons.

The actual picture is generated by finding a solution to the Constraint Satisfaction Problem and modified icons in the correct positions and the relationship markers in the correct positions. Each window is drawn with its corresponding window solution.

Figure 3.3 lists the components and the processes of pictorial representation.

In summary, the concept makes use of a logical form as the input. The visual objects, and the relationships (primarily verbs and prepositions) are identified within the logical form. Then each of the objects are assigned an icon, with their associated size, while the relationships define the Pictorial Representation Windows (PRWs), and placement or modification of the icons within the PRWs. The placement and modification of icons are defined by a set of constraints. At this point, a constraint satisfaction problem exists consisting of a set of PRWs, each with its own set of icons and a set of constraints acting upon those icons. Finally, the constraints are solved, and the solution is graphically displayed. A complete description of these processes, with examples, is given in the next chapter, "The System Design."

3.3 How good is the Pictorial Representation?

3.3.1 Goal of a Good Pictorial Representation Scheme

Several factors are needed for a "good" pictorial representation. Primarily, the representation needs to be rich enough to represent all that it should represent, at the same time, that it doesn't represent more than it should, and most of all, it needs to be understandable. In other words, the goal of a good pictorial representation scheme is to be:

- **Input**
  - Logical Form
- **Knowledge from Logical Form**
  - Visual Objects
  - Relationships between Visual Objects
- **Pictorial Components**
  - Icon representation of each Visual Object
  - Size of each Icon
  - Pictorial Representation Window (PRW)
- **Operations on Components**
  - Placement of Icons within PRW
  - Modification of Icon (add/del pieces within icon)
  - Emphasis of Icon (reverse video)
  - Inclusion of a New Icon
- **Output**
  - Set of PRWs
  - Each PRW has a set of constraints (operations) acting upon its components

Figure 3.3: Components and Processes of Pictorial Representation

- 1) Capable of pictorially representing even subtle textual differences to the viewer;
- 2) Minimize false implicatures;
- 3) Understandable to a viewer with limited or no training.

Related to the first two criteria, are the issues of ambiguity and vagueness. A pictorial representation needs to be able to represent textual ambiguity. In addition, textual information is often too vague to generate a specific pictorial representation of it. This can lead to false implicatures.

Ambiguity and vagueness, as well as understandability are important issues to a pictorial representation scheme. They will be discussed in the next two sections.

3.3.2 Handling Ambiguity and Vagueness

A excellent way to look at the representation is to ask "is it ambiguous when it shouldn't be?", "can it capture ambiguity when it should?" and specifically "can it represent pictorially all of the possible meanings of a textual sentence that is ambiguous syntactically and semantically?"

I believe there are four types of ambiguity that are particularly relevant to pictorial representation:

- 1) Lexical Ambiguity
- 2) Syntactic Ambiguity
- 3) Semantic Ambiguity
- 4) Pictorial Ambiguity

Lexical ambiguity is discussed in Chapter 8, "Ambiguity and Vagueness." One type of lexical ambiguity is between categories, such as 'rose' which is a verb in sentence 8, or as 'rose' as a noun shown in sentence 9. This type of lexical ambiguity is not of significant interest to the pictorial representation problem, as it is resolved by the natural language system.

(8) The price rose to over three pounds.

(9) Anna was given a rose by her boyfriend.

The real topic that needs to be addressed in this thesis concerning lexical ambiguity, is the lexical ambiguity within the same category. This can be a problem pictorially, when associating an icon to a noun. For example, given the noun "bank", it could mean a 'bank' with money as or a 'bank' next to the river. Each would have a different icon representation.

Syntactic and semantic ambiguity should be adequately captured in the pictorial representation. One of the original goals of this text–to–pictures system was to display all of the possible intended meanings of a textual sentence that were syntactically and/or semantically ambiguous. Given sentence 10, there are several interpretations. The pictorial representation must be such, that it can *represent each* of the interpretations. It does this by presenting a separate picture for each reading.

(10) Carla saw the man on the hill with a telescope.

The syntactic ambiguity of whether the 'on' preposition is attached to a 'man' or 'Carla' will result in two different logical form descriptions. The two logical forms will yield two different pictorial representations. The semantic ambiguity of the preposition 'with' is indicated by different definitions of the word in the CLE lexicon. For example, the 'with' preposition could mean *with_Accompanying* or could mean *with_Instrument*. For each of different syntactic descriptions for a sentence, a unique logical form for each different meaning of the word 'with' is generated, thus a large set of logical forms could be generated for one sentence. The goal of the pictorial representation is to show a different and unique visual scene for each different and unique logical form corresponding to a same sentence. The pictorial representation must capture the subtleness of the syntactic and semantic differences between the logical forms.

Pictorial ambiguity is very interesting and critical to the test of a good pictorial representation. A pictorial representation can remove or reduce the other three types of ambiguity, however an additional ambiguity can be introduced. A certain representation

may be interpreted by several viewers in different ways. An icon may have different meanings to different viewers. This is the unwanted side-effect of using pictorial representation to remove the other types of textual ambiguity. To minimize pictorial ambiguity, some training of the viewer should be done. The training is implemented, as will be shown in the following chapters, with a pictorial definition window, which will assign a description to each icon. Additionally, using several pictures to disambiguate an ambiguous textual sentence, in association with the original text, can minimize pictorial ambiguity, because the icon to noun ambiguity is reduced to the list of nouns within one sentence, and only the differences between the pictures need to be interpreted correctly.

Quite different from pictorial ambiguity, in fact, almost an opposite situation, is when too much information is introduced into the pictorial representation. This situation can be described more precisely, as when false information is introduced. Precise information is usually required to generate a picture. However, sometimes the information contained in the text is **vague**. Therefore, some assumptions are needed to make the vague information more precise. The problem arises when an incorrect assumption is made, which could lead to false implicatures.

Given that a "glass is on the table" one can see exactly where it is located on the table in the picture. This is a problem because of the overspecificity of pictures compared with text. Ideally, a pictorial representation should have the capability to *imply* to the viewer that the exact position of the glass is not known, unless it was specifically stated in the text.

Primarily, a good pictorial representation reduces lexical, syntactic, and semantic ambiguity, while minimizing introduction of pictorial ambiguity or false implicatures. This is not enough for the representation, as it must also be *understandable*.

3.3.3 Understandability Requirement

The pictorial representation should be able to be quickly *understood* by an observer for it be of much value. In the language domain, words have definitions given in a dictionary, and the grammar is clearly defined and understood by all that can converse in the language. However, pictures do not have a precise definition of icons nor is there an official standard convention of presenting icons. This does not mean that some heuristics

do not exist that will yield pictures that are more easily understood than others. Also, some training of the user must be assumed. Upon the first display, the viewer may not be certain that a location of icons represents what the system intended. However, based upon the small number of subjects who used the system, after relatively few displays, the viewer quickly became accustomed to the association of a certain icon to a certain object. Usually, the *meaning* of a location of icons was discerned after looking at all of the pictorial representations for a sentence. For example, in sentence 10, the case of the telescope being used as an instrument, or the case where the telescope is accompanying something, is shown differently. By looking at the differences between the two pictures, the meaning became clear.

But does this affect the *pictorial representation*?. The principal problem is the interpretation of picture. Some argue against the use of icons. "The primary problem with the possibility of an iconic canon is that icons articulate no discrete subject–predicate relations, contain much irrelevant information, and embody no explicit logical structure. Hence, there is no well–defined pictorial grammar or vocabulary and no obvious rules of interpretation" [Summary of Fodor83].

But the fact that pictures lack syntactical features of the sort that linguistic representations share poses no real impediment to their formal disambiguation. The idea I have suggested is that a pictorial representation has a compositional character of a set of objects, features, and relationships. This leads to pictures that do in fact represent to the viewer the intended meaning of the textual sentence.

The concept of pictorial representation also should be bounded. By this, I mean that only a limited level of *understanding* can be immediately depicted by the representation. However, the representation is rich enough that higher levels of *understanding* could be deduced by further intelligent processing (i.e. expert systems, or the user).

A definition of *understanding* is presented. It was motivated by a definition of understanding in comparative studies of the development of this process in weak-sighted school children of both normal and abnormal intelligence, which identified five levels of understanding narratives [Golvina 74]. The levels of understanding used within the pictorial representation are:

1) The identification of icons to represent objects within a sentence;

2) establishment of relationships among objects determined directly from the syntax and semantics of the text, to include prepositional and verbal relationships;

3) simple employment of naive physics relationships, techniques to reduce false implicatures, and techniques to provide an aesthetic picture;

4) establishment of simple cause–and–effect relationships, pragmatics, object usage, and movement;

5) establishment of a logically consistent system of cause–and–effect relationships, full pragmatics, inferencing used to determine plausibility of representations, time sequencing and representation of discourse.

The pictorial representation scheme attempts to represent information at the first three levels of understanding. Objects are represented by icons. Also the establishment of relationships between objects is discerned from the text and represented by location of icons, modification of icons, or inclusion of new icons. Naive physics and other simple heuristic techniques are used to provide a minimal understanding from the pictorial representation.

Causal relationships, pragmatics, object usage, and movement, although they would provide a more interesting representation, are beyond the scope of this thesis. This is also true for the highest level of understanding of discourse representation, full causal relationships, time sequencing, et cetera.

3.4 Restrictions and Limits of the Concept

First, this thesis is about *Pictorial Representation of Text* and not a conversion process that translates in both directions from Text and Pictures.

What would be the "reverse" process of this thesis–Textual Representations of Pictures? At first glance this may seem plausible. However, this seems to define another process and certainly involves different input data. Pictures can include photographs, maps, drawings, et cetera. Therefore, a more adequate description of the reverse translation of the process described in this thesis, would be to develop a textual description of a pictorial representation. Such a process would only be useful if another system could generate a

pictorial representation from an actual scene, photograph, or map. Such a system would be the visual processor analogous to the CLE text processor that can operate on raw text. I did not have access to any system that could generate a pictorial representation given a photograph. Therefore, the process of generating a pictorial representation from a textual input was considered as the only process that should be tackled for this thesis.

The second limitation can also be illustrated by looking at the reverse problem. In generating text to describe a picture, one critical process is simply choosing a word to textually represent a visual object. Jerry Fodor describes the problem of looking out of a window and seeing "a lady walking a dog." He states that the viewer could have also described the situation as "a lady walking an animal" or even "a lady walking a silver-gray, miniature, poodle bitch" [Fodor 83] (p. 96). He gives a *cluster of psychological properties* that tend to describe a preference that a viewer would use in textually describing a visual scene.

However, one may say that I still have the same problem–if the input has the word 'alsatian' then I may want to produce an icon that the user interprets as 'dog' or 'guard dog' or 'Fido,' say. This is true. Therefore, a limitation is required, which is to allow the user to define the icons. This makes the system much more manageable (and useful). If the user defines the 'dog' icon to represent 'poodle,' 'alsatian,' and 'german shepherd,' then the system will display the same icon when encountering these three separate words. This type of knowledge is rather domain specific and should be separated and left to the user to define.

A third limit of the research is one of granularity. Since the topic chosen is rather broad, a general mechanism is shown. For example, a technique is shown how spatial prepositions can be described via a set of pictorial primitives and constraints. But this does not imply that I have the perfect set of primitives or constraints for a given preposition. Someone will certainly have a better definition of the preposition 'on' than the one I have used (i.e. some people are conducting their entire PhD thesis on a single preposition [Vieu 91]). However, this limitation does not restrict other pictorial definitions of certain linguistic expressions. On the contrary, an advantage of this system is that others can experiment with their pictorial definitions of linguistic expressions by using a system based upon this concept.

The fourth limit is a functional one–that the process handles a single sentence as input. Multiple sentences that contain anaphoric references and that add or more fully define/describe objects increases the complexity of the process. This can be even carried further to a data fusion process of pictorial displaying relevant data/information from several textual sources. The data fusion process would allow a person to find patterns in the data among several input sources. Design issues of such a system are described in Chapter 9, "Possible Applications." However, before attempting to build a system to do that, first the basic properties of pictorially representing one sentence must be understood.

The fifth limit is that certain linguistic expressions are not pictorially represented in the working system. The primary reason for this limitation was to reduce the scope of this broad project. In addition, methods of pictorial representation are not apparent for some types of linguistic expressions.

Interrogatives, and imperatives are not handled. The system works only with declarative text. This does not mean that a pictorial representation could not be somehow be generated for interrogatives or imperatives. The goal was to generate pictorial representations for declarative statements. In future work, it may be possible to add some marking to change the representation to show an interrogative or an imperative–almost in the same way punctuation is used. However, that is outside the scope of this system.

Other linguistic limits are that the system does not handle: the definiteness of an object(no distinction between definite and indefinite objects), comparators, adverbs, adjectives, or universal quantifiers. Although, the system does not represent these expressions, they are discussed further in Chapter 5, "General Expressions."

Finally, it must be said, that no claim is made that this process is the cognitive process used by humans to generate a mental image of text that they have just read. Rather, this concept of pictorial representation is offered as a possible means to build a working system to generate a pictorial representation of text.

3.5 Advantages of the Pictorial Representation Concept

Given the previous set of limitations, this allows several advantages to be realized. These advantages include the building of a working system, a technique for resolving textual ambiguity, a technique for rapid interpretation and integration of data, a test–bed system to allow experimentation of pictorial definitions of language components (e.g. prepositions), and notably, a principled approach to the translation process of text to pictures.

The working system is a definite advantage. Although, it may not have been required, it does lend some extra credibility to an approach. Further, it allows for experimentation, and quicker maturation of ideas.

Several researchers, for example [Cortes 89] and [Wahlster *et al* 91b], have mentioned the advantage of combining natural language with graphics as it gives two communication mediums, but also allows the two to be used in parallel, hopefully reducing the ambiguity by showing two different representations–textual and pictorial. There are many instances when the textual ambiguity and pictorial ambiguity do not overlap. In these cases, using both representation systems, the overall ambiguity is reduced. Pictorial representations could be used to accompany natural language systems to resolve ambiguity for non–linguist users. Such a system is shown in Chapter 8, "Ambiguity and Vagueness."

Using this concept as an experimental test system to determine specific pictorial definitions of language is useful. In the next chapter, "The System Design," I show how my system was purposefully designed so that some modules are not accessed by the user (e.g. the set of pictorial primitives), where other modules can be modified by the user (e.g. a pictorial definition of a certain preposition). If a particular definition of a language component is too relaxed or too restrictive, it will be immediately evident to the user by looking at the resulting pictorial representation. The user then could modify their set of language component definitions, until the appropriate pictorial representations were generated.

Finally, this concept allows a definition of the translation process from text to pictures to be described in one technique, namely via a set of constraints. Much of this thesis describes particular linguistic components and how they are translated via constraints to form a pictorial representation.

Chapter 4

The System Design

This chapter describes the system that pictorially represents text, and specifically describes the system's design. Basically, the system consists of a conversion of text into a intermediate representation scheme, followed by a picture generation process. The system contains several components. For each component, I describe its function, expected input and sample output, and the operation of the process.

It is important to note that although the entire Text–To–Pictures System (TTPS) is described, this thesis deals primarily with the picture generation process. The components that convert the natural language text into a representation scheme are the Core Language Engine and are included only to show a complete system. The CLE processes are described in detail in other reports [Alshawi *et al* 88], [Alshawi *et al* 91]. In effect, they are treated as "black boxes" in this thesis.

4.1 Top Level System

The goal of this system is to take text and produce a picture that "represents" the intended meaning of the text. From a top–level view, the system converts the text into a logical form representation. Then each of the "objects" contained in the logical form representation is represented by a pictorial object (i.e. an icon). Then, these icons are placed in an imaginary space via a set of pictorial constraints. After all of the constraints are determined, the system attempts to solve the constraint satisfaction problem. If a solution is found, the objects are drawn within a pictorial representation window.

The process just described places objects within a pictorial representation window (PRW) using a spatial representation. However, each spatial PRW also has an associated temporal PRW, showing the time of an event or state. The associated temporal PRW is displayed directly beneath the associated spatial PRW on the graphics display. Sometimes more than one spatial PRW is needed to represent the sentence. An example of a multiple PRW representation was shown in figure 3.1. In that case, each PRW has its own set of constraints and a solution is generated for each PRW.

Notably, the constraint generation is critical to the system. Each sentence is represented by a large set of pictorial constraints, many of which are provided by the LF representation. Two example sources of pictorial constraints generated directly from the LF representation are prepositions and verbs–they provide constraints to pictorially describe relationships between objects. Other pictorial constraints are needed to handle naive physics, to provide missing information, to prevent false implicatures, and to simply provide an aesthetic picture.

Examples of two sources of pictorial constraints that are generated directly from the sentence are presented–specifically from the prepositions or the verbs in the sentence.

(11) The cup is on the table.

(12) The man saw the dog.

In sentence 11, the preposition *on* indicates a relationship between two objects–the cup and the table. This relationship can be described as a set of constraints. In English, the set of constraints may be: the cup is above table, the cup is not left of the table, the cup is not right of the table, and the cup is touching the table.

In sentence 12, the verb *saw* also indicates a relationship among objects–the man, the seeing event, and the dog[1]. This relationship will generate a set of constraints among the pictorial objects of the man, the seeing event, and the dog. Assume a pictorial representation for the seeing event, such as an icon that looks like an eye[2]. Assuming

[1] I am not claiming that all verbs are like this example, but I do want to show that the verb involves a relationship among some objects. In some cases, the verb is an event. In others, it is simply stative or indicates that some object exists. I am using the term *objects* as pictorial objects and not as a linguistic term.

[2] The user is able to define his or her own set of icons.

this, the set of pictorial constraints in English may look like: the seeing event is near the man, the seeing event is far from the dog, and the seeing event is between the the man and the dog.

There are other forms of constraints that are not generated directly from the sentence. The **naive physics module** generates some placement constraints. An example is the role that gravity plays in the picture generation process. In sentence 11, the cup is *on* the table. If 'on' is defined as "X is *above*, *not left of*, *not right of* and *touching* Y," this would seem to work for sentence 11. The precise definitions of such terms as "above", "left", "right", and "touching" are defined later. For now, one should think of the intuitive meaning of these terms.

(13) The spider is on the table.

For sentence 13, the previous definition of 'on' may be too restrictive. It would require the spider to only be "on" the top of the table. However, the spider could be "on" the side of the table. If one assumes there is a constraint called gravity that requires that one object must rest upon another object, then the constraints that are generated by the preposition 'on' are relaxed to only require "touching," along with the gravity constraint. This is true for most objects in real life, and works for "the cup on the table" example. However, some objects, such as a spider can defy gravity. In this case, the gravity constraint is not applied, and only the "touching" constraint is applied.

Before a picture can be generated other constraints must be generated. For example, in generating a picture for sentence (11), some specific information must be added to actually give the cup a location in relation to the table. Where does one place the cup *on* the table? To draw a picture one needs an exact location. However, the sentence alone does not give an exact description. So a **missing information module** will help clarify this problem. These modules are described later, but this should demonstrate that there are several types of knowledge required to generate a pictorial representation for a sentence of text.

An attempt was made to make this Text–to–Pictures System as modular as possible. One reason was to demonstrate the many different types of knowledge that are required to generate a picture. Modularity is also especially important in the use of the sys-

tem in experimentation. A user may be interested in developing naive–physics constraints [Hayes 79], or perhaps in the development of definitions of spatial prepositions [Borillo & Borillo 90], [Herskovits 86]. The user would like to develop his or her own constraints without worrying about the operation of the rest of the system. A final reason that modular design is important, is to separate domain–independent knowledge from domain–dependent knowledge, which increases portability [Grosz 83], [Martin *et al* 83].

The following is a diagram of the system.

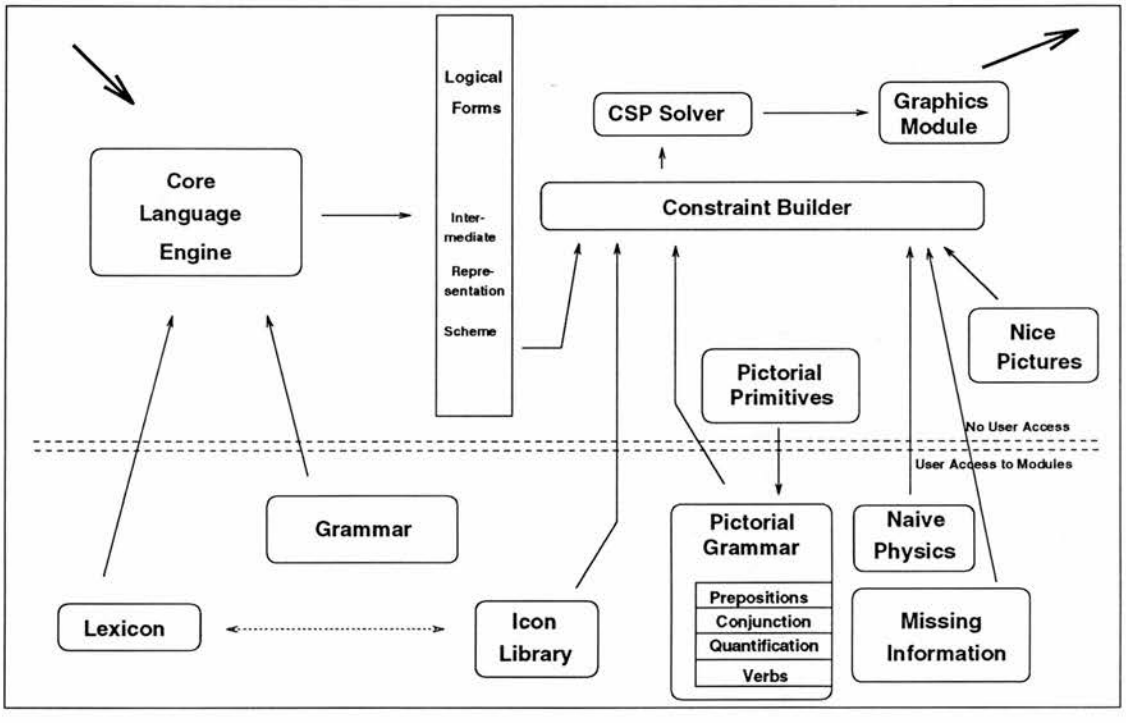


Figure 4.1: The Text–to–Pictures System Design

The system contains the following modules:

a) Core Language Engine

b) Constraint Builder

c) Constraint Satisfaction Problem Solver

d) Graphics Module

with the following Knowledge Sources for the Language Engine:

e) Grammar

f) Lexicon

and the following Knowledge Sources for the Pictorial Generation Process:

g) Pictorial Primitives

h) Pictorial Grammar (several sub-modules)

i) Naive Physics

j) Missing Information

k) Nice Pictures

l) Icon Library

and the data base of the logical form representations also merits mentioning:

m) Intermediate Representation Scheme

A simple description of each component follows:

4.1.1 A – The Language Engine

The language engine used in the Text–to-Pictures System is the Core Language Engine (CLE) developed by SRI/Cambridge. The language engine reads in a sentence and yields a logical form representation for the sentence. The engine is actually made of several components: a syntactic parser and components that handle scoping of quantifiers, conjunction ambiguity, anaphora problems, preposition attachment, as well as some case assignment.

The parser yields a syntactic parse tree to represent the sentence. The system should produce a separate parse tree for each possible syntactic representation of a sentence. If a sentence has several possible syntactic readings because of ambiguity (e.g. unknown prepositional phrase attachment), the system should yield a parse tree for all of the possible syntactic readings.

Other CLE modules take the parse tree representation and yield a logical form representation. Sometimes these modules cannot resolve the intended meaning. It could be because of an actual ambiguity. In this case, more than one logical form representation is produced–other processes need to resolve the actual meaning intended by the speaker/writer. It could also be that modules themselves require more information to determine a logical form representation. In these cases, a "quasi–logical form" representation is used, or several logical form representations are given–one for each possible meaning [Alshawi & van Eijck 89], [Alshawi 90].

4.1.2 B – Constraint Builder

This module builds a set of constraints between objects. This set of constraints is a pictorial description in an imaginary space. Later the positions of objects are mapped to a pictorial representation window giving a visual scene. The process involves determining what are the objects, and building or selecting all of the constraints upon these objects and the constraints among the relationships of these objects. Constraints are specified in the knowledge modules, such as Sentence Constraints, Naive–Physics Constraints, Missing–Information Constraints, or Nice–Picture Constraints. The constraints are defined via a set of pictorial primitives. This module also handles icon modification (e.g. highlighting a icon), and placement of multiple pictorial representation windows.

4.1.3 C – Constraint Satisfaction Problem Solver

The Constraint Satisfaction Problem (CSP) Solver takes the set of constraints and attempts to find a solution. The problem is solved by using a coordinate space representation and searching for locations that objects can occupy that satisfy the given constraints. The final solution is a set of objects with appropriate positions on the grid or coordinate space. This process can be very costly in computational terms depending on *how* the search for solutions is conducted.

4.1.4 D – Graphics Module

This procedure takes the output of the CSP solver, which is a list of the objects with a location on a grid for each object, as well as associated icon for each object, and draws it.

The graphics procedure also involves scaling the imaginary coordinate space coordinates to coordinates for the picture (or graphics screen). The objects are drawn via a set of graphics commands within GM (Graphics Manager) in SICSTUS Prolog [Carlsson 91b], [Carlsson 91a].

4.1.5 E,F – Knowledge Sources for the Language Processing

The Language Engine requires a grammar and a lexicon. These modules are shown separately in figure 4.1 because of their fundamental importance to a language engine and the fact that these modules can be modified by the user. In fact, a special program called VEX [Alshawi *et al* 91], [Carter 92] allows the modification of the CLE lexicon by the user. The user also can define a set of sortal restrictions if he chooses. Sortal restrictions are used by the semantics resolver modules to reduce some of the syntactic representations that are not plausible [Katz & Fodor 68].

4.1.6 G – Pictorial Primitives

There are several knowledge sources that contain constraints for the pictorial generation process. At the lowest level these constraints are defined in terms of a set of primitives. Examples of such primitives are: above, below, left-of, right-of, et cetera. These examples are two-place predicates that describe the relationship between two objects in terms of an imaginary coordinate space. For example, in a two-dimensional coordinate space, with the Y axis representing up and down, the primitive *above* is described as object one having a greater Y value than object two. It should be stated that some primitives are three– and four–place predicates. A detailed description of the primitives is given in the section 4.2, "Pictorial Representation Approach."

4.1.7 H – Pictorial Grammar Module

The pictorial grammar or sentence constraint module uses the logical form representation of the sentence to generate pictorial constraints. The top–level of the pictorial grammar module looks at the logical form representation of the sentence and identifies the *objects*, and also identifies what parts of the logical form are *relationships* among the objects. The pictorial grammar module also contains several sub-modules. There is a preposition

sub–module, one for verbs, another for quantification, et cetera. Each sub–module looks up the pictorial definition for the appropriate word. When a *relationship* is identified in the logical form, the appropriate sub-module will translate the logical form *relationship* acting upon its objects, into a set of pictorial constraints acting upon the pictorial objects(icons). The pictorial definition for a certain word can be modified by the user by editing the pictorial definition file. Earlier, examples of the how the preposition and verb sub-modules generate a set of pictorial constraints were shown. A detailed description on this subject is given in the section 4.3.2, "Pictorial Grammar Module."

4.1.8 I,J,K – Other Knowledge Sources for the Pictorial Generation Process

The other constraint modules include one for naive physics, one to supply missing information, and another to produce nice pictures. Each of these modules provides constraints to the picture generation process that are not provided by the pictorial grammar (sentence constraint module), but are required to generate a pictorial representation.

The naive–physics module provides a small set of constraints that act upon all or most of the objects. The gravity constraint is an example of naive physics. Another example of naive physics is that two objects cannot occupy the same space at the same time. The missing–information module adds information that a picture would need but that the sentence did not provide, such as the location of the cup on the table. This module also tries to reduce false implicatures by preventing certain placements of objects that could be construed in another way. The nice–pictures module adds constraints that yield a more pleasing picture. By using these modules, constraints are added to the constraints generated by the pictorial grammar (constraints produced from the sentence) to form one long list of constraints. These modules are described in detail in section 4.3, "Pictorial Representation Generation."

4.1.9 L – Icon Library

This is a library that stores a pictorial definition of objects. Each icon's representation and size coordinates are stored in this library. My system does not make use of color, but in a future system, the color information would be stored in the icon library. It also

contains some feature information that is used to determine whether constraints apply or not to this object. This is analogous to the lexicon's function to the language engine. When the constraint builder module receives the object list, it assigns a pointer to the icon in this library. In the final stage, the graphics module will draw the icon.

The icon library can be modified and added to by the user. If a user does not like an icon representation for a certain object, they can change it to their preference. The icon library differs from the lexicon, in that there is generally more consensus about definitions for words via dictionaries, than there is for pictorial definitions.

4.1.10 M – Intermediate Representation Scheme

The intermediate representation scheme consists of a set of logical forms. There may be several logical form representations for one sentence because of the several possible meanings of the sentence. The logical forms are stored in a Prolog data base. The data base is shown in figure 4.1 as an integral part of the text–to–pictures system for two reasons.

First, if this system is expanded to handle sections of text rather than only one sentence, it will require the use of the data base which contains logical forms for each sentence of the discourse. Second, in my implementation I did not have both the Core Language Engine and the pictorial representation generator on the same computer. This was because of licensing and contractual agreements. Therefore, the intermediate representation scheme was truly the interface between the language engine and the pictorial generation process.

The CLE logical form representation was fully described in section 2.3.3, "CLE Logical Form Representation Scheme."

4.2 Pictorial Representation Approach

The pictorial representation approach is built upon the idea that a picture can be generated that represents to the viewer the intended meaning of the sentence. Assuming that each object can be represented by an icon, then the problem is reduced to placement of the icons. A two-dimensional grid is used to represent an imaginary space to place the objects upon. A set of constraints will restrict the placement of these objects. In

other words, the pictorial representation generation process is defined as a constraint satisfaction problem.

This section will describe: what is a constraint satisfaction problem, how to formulate the pictorial generation process (or icon placement problem) as a constraint satisfaction problem, and last an example of actually placing the icons using this technique. To simplify the explanation, only one pictorial representation window is used, and the icons all have the size of one by one units. The actual system makes use of multiple windows and of size. That will be discussed in section 4.3, "Pictorial Representation Generation."

First, the definition and description of a constraint satisfaction problem.

4.2.1 Constraint Satisfaction Problem

The structure of a **Constraint Satisfaction Problem** (CSP) can be defined as a triple consisting of [Bodington & Elleby 88]:

- a finite set of variables
- a finite set (*domain*) of candidate *values* for each variable.
- a finite set of *constraints* on the values that various combinations of variables can be assigned simultaneously.

Which can be written as:

- a set of variables $\{v_1, v_2, ..., v_n\}$
- each variable has an associated domain, $D_i, i = 1, 2, ...n$
- a set of constraints relating the allowed values of each variable $\{g_1(v_1, v_2, ..., v_n), g_2(v_1, v_2, ..., v_n), ..., g_m(v_1, v_2, ..., v_n)\}$

In this definition, some clarification and further specification is needed.

- *Assignment* is the association of a variable with a value from its corresponding domain.

- *Domain* - According to the structure of the Constraint Satisfaction Problem, there is no particular restriction on the domain of each variable. However, in order to apply several search techniques, the domain of each variable is often required to be a finite set of discrete values, as is the case in the working system implemented for this thesis.

- *Constraints* - extending the basic Constraint Satisfaction Problem, it is possible to further define the constraints as having two types of priorities:

  - Inter–priorities (relative importance of constraints). Sometimes it is not possible to find a solution satisfying all the constraints. In order to overcome this problem it is usual to assign priorities to each constraint. If necessary, constraints with a lower priority will be first relaxed.
  - Intra–priorities (relative importance of different values for a constraint). A single constraint may itself have a weighted preference. While solving for this constraint, an attempt is made to choose the best among several choices. These choices can be thought of a sub-constraints.

  An example would be for the constraint *far–from*. Object A may have to be *far–from* Object B with a priority of 5 (high). This constraint has a higher *Inter–priority* than a constraint that Object C *touch* Object D with a priority of 1 (low). However, the *far–from* constraint may be defined as: (d = distance between the objects) $d > 10cm$ with high intra–priority of 5, and $d > 30cm$ with an intra–priority of 3. This can be thought of as: the distance must be at least 10 cm apart, but at least 30 cm is preferred.

- *Solution* - depending on the particular problem, one of the following situations may occur:

  - only one solution is required - the first solution
  - several alternative solutions are required
  - "the best" solution (according to some criteria) is required

4.2.2 How to Formulate the Icon Placement Problem as a Constraint Satisfaction Problem

In order to formulate the icon placement problem as a constraint satisfaction problem a preprocessing phase is required. This phase will identify:

- each icon to be placed on the screen (variables)
- the inter–relationships among the icons (constraints)
- inter–priorities among the constraints
- intra–priorities within the same constraint,
- general criteria to place the whole picture on the screen
- based on the domain, a granularity of the grid

The icon placement problem can be defined as:

> Try to assign each icon to a point on the grid ensuring that all the constraints are satisfied.

But this definition needs to be clarified. First, to actually solve this problem (as it will later be shown) the size of the grid or screen is important. To ease the solution process, an imaginary grid is used rather than coordinates on the screen. In the following example, I will show a two-dimensional grid with X and Y coordinates. The grid can vary in size. However, the size must be determined before the constraint satisfaction problem can be solved. Also, for this definition it should be noted that each icon will take up only one point. Therefore, each icon will have the same size. In the actual system, icons do have varying size. This assumption is used throughout the example to simply the explanation.

The structure of the constraint satisfaction problem is described.

- Variables - each icon has an associated 2-D variable. This variable represents the possible location of the icon and it is identified by two coordinates (x,y). By default, the domain of each variable is the set of points on the grid which represents the screen. Of course this default can be changed using constraints.

- Constraints (and Pictorial Primitives) - There are two types of constraints: relative and general.

  Relative Constraints define relationships among the icons. These constraints are made up of combinations of pictorial primitives. A small set of pictorial primitives (inter–relations) are defined, which will be used in an upcoming example. Considering icons A, B, and C, the small set of primitives would be:

primitive	*description*
above(A,B)	A is above B
left-of(A,B)	A is to the left of B
right-of(A,B)	A is to the right of B
touching(A,B)	A touches B
near(A,B)	A is next to B
far-from(A,B)	A is far from B
between(A,B,C)	A is between B and C

  – The "not" relationship can apply to each of these seven constraints. Therefore, *Not(near)* and *far-from* would not be the same relationships.

  There are many other relative constraints or pictorial primitives that are used in the actual system. They are described in later chapters. For example, these seven constraints are a subset of the spatial expressions pictorial constraints. The full set of spatial constraints is described in Chapter 6. This minimal list is only given for purposes of the illustrating the upcoming example, in the next sub–section.

  General Constraints work upon the overall picture. An example of some general constraints are the nice–pictures and naive–physics constraints that deal with: an icon must be on screen, vertical and horizontal centering, use of margins, preventing overlap, gravity, et cetera. These constraints will have priorities associated with them so that some violations of these rules can occur to satisfy more important constraints.

- Priorities - priorities are attached to each constraint as a mechanism to allow relaxation. Constraints are ranked according to different priorities. If no legal solution can be achieved satisfying all the constraints, relaxation will be adopted. Constraints with the lowest priority are relaxed first. In the upcoming small example,

all of the "relative constraints" are assumed to have the highest priority. General constraints, not as important as relative constraints, can have a lower priority. If necessary, they will be relaxed.

4.2.3 A Simple Example

In this simple example, a set of constraints is generated for the following sentence, the constraints are solved, and then the solution is graphically displayed. In the real system, the constraint list is larger, but for this example, only a minimal list of constraints is shown.

(14) The cat is on the table.

This example will concentrate on the constraint satisfaction problem. The next section steps through the actual system, including the use of logical forms. For now, assume that the objects have been identified as 'cat' and 'table', and a relationship between those objects of 'on.' Each object is assigned a variable, which can take a grid assignment.

- Icons / Variables
  - cat $\Rightarrow$ A(X_A,Y_A)
  - table $\Rightarrow$ B(X_B,Y_B)
- Size of Grid - The size of the grid used is five by four[3]. Therefore the domain is twenty points for each variable.
- Priority Scheme - A priority scheme between 1 and 5 is used. 5 is the highest priority and 1 is the lowest.
- Constraints - The Relative Constraints result from the preposition 'on', which is defined in terms of the pictorial primitives as:

```
on(A,B)  <=>  above(A,B)      and
              not left(A,B)   and
```

[3] This size of the grid is only for the example. The grid size could be bigger. In many of the sentences that are displayed by the actual text-to-pictures system, a grid of 200 by 200 or larger is used.

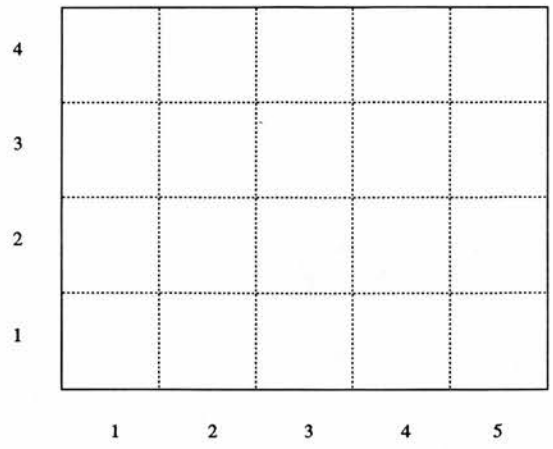


Figure 4.2: Example Grid

```
not right(A,B)  and
touching(A,B)
```

Where the primitives are defined as:

$$above(A,B) \Longleftrightarrow Y_A > Y_B$$
$$left(A,B) \Longleftrightarrow X_A < X_B$$
$$right(A,B) \Longleftrightarrow X_A > X_B$$
$$touching(A,B) \Longleftrightarrow X_A \in \{X_B - 1, X_B, X_B + 1\} \cap$$
$$Y_A \in \{Y_B - 1, Y_B, Y_B + 1\}$$

and given that:

X_i is the X coordinate of icon i, (i = A, B)

Y_i is the Y coordinate of icon i, (i = A, B)

All of the relative constraints are given the highest priority of 5.

This yields the following relative constraints:

```
above(cat,table)
not left(cat,table)
not right(cat,table)
touching(cat,table)
```

or more specifically (by applying the pictorial primitive definitions):

$$Y_{cat} > Y_{table}$$
$$X_{cat} \geq X_{table}$$
$$X_{cat} \leq X_{table}$$
$$X_{cat} \in \{X_{table} - 1, X_{table}, X_{table} + 1\} \cap Y_{cat} \in \{Y_{table} - 1, Y_{table}, Y_{table} + 1\}$$

The following general constraints will be used in the example:

- all objects/icons will be on the grid. This is by definition of the problem.
- overlap of two icons is prohibited. (priority 4).
- horizontal centering (priority 3).
- vertical centering (priority 2).

Therefore, this means that constraints acting on the icons have the highest priority. The overlap problem is solved and then the centering problem is solved, with horizontal centering having higher priority than vertical centering [4].

The Overlap Check and Horizontal and Vertical Centering are given.

Overlap check:

$$overlap.check(A, B) \Longleftrightarrow X_a = X_b \quad \wedge \quad Y_a = Y_b.$$

Horizontal and Vertical Centering:

$$horiz\_center \Longleftrightarrow \; | \, (X_g - Xp_{high}) - (Xp_{low} - 1) \, | \; \leq \; 1$$
$$vert\_center \Longleftrightarrow \; | \, (Y_g - Yp_{high}) - (Yp_{low} - 1) \, | \; \leq \; 1$$

Where:

X_g = upper limit of grid horizontally
Y_g = upper limit of grid vertically

Xp_{low} = lowest x value of the picture
Xp_{high} = highest x value of the picture
Yp_{low} = lowest y value of the picture
Yp_{high} = highest y value of the picture

[4] The priority scheme will have no influence unless some constraints cannot be solved. In that case, it is the lowest priority constraints that are first *relaxed* in an attempt to find a solution.

This yields the following general constraints:

```
not overlap(cat,table)
horizontally-centered
vertically-centered
```

The not operation applied to the *overlap* constraint, causes a disjunctive constraint to be used[5].

$$not\ \ overlap(A, B) \iff X_A \neq X_B \ \vee \ Y_A \neq Y_B.$$

Given that the grid size is five by four, and applying the definitions of the constraints in terms of X and Y values, the general constraints are:

$$X_{cat} \neq X_{table} \ \vee \ Y_{cat} \neq Y_{table}$$
$$| (5 - X_{ph}) - (X_{pl} - 1) | \leq 1$$
$$| (4 - Y_{ph}) - (Y_{pl} - 1) | \leq 1$$

where:

$$X_{ph} = \text{greater}(\ X_{cat}, X_{table})$$
$$X_{pl} = \text{least}(\ X_{cat}, X_{table})$$
$$Y_{ph} = \text{greater}(\ Y_{cat}, Y_{table})$$
$$X_{pl} = \text{least}(\ Y_{cat}, Y_{table})$$

Based upon the total set of constraints (the relative constraints and the general constraints), the legal solution is:

Icon A = (3,3) and Icon B = (3,2)

Figure 4.3 represents the solution. The cat icon (Icon A) is drawn in position (3,3) and the table icon (Icon B) is drawn in position (3,2).

The basic process was described. The following section describes how the actual system works in detail.

[5] The *not overlap* constraint is the only constraint that must be defined disjunctively. This is discussed in detail in Chapter 6, "Spatial Expressions."

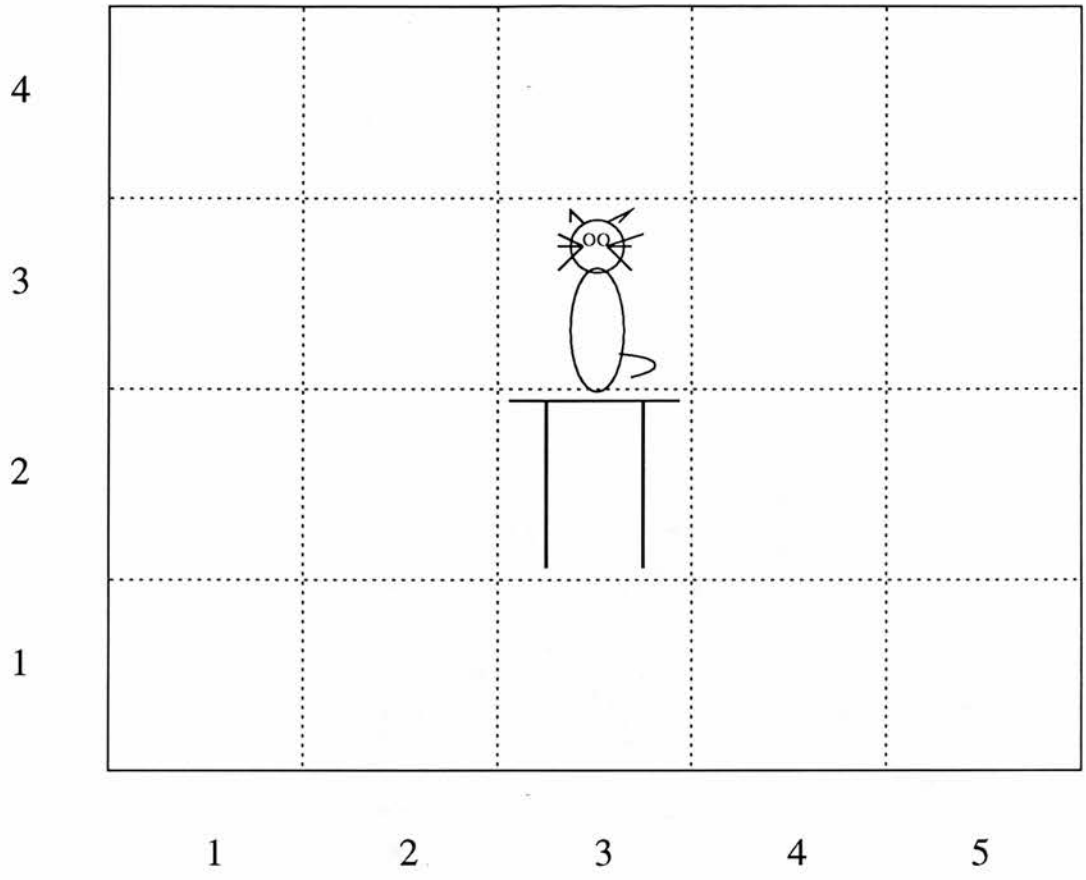


Figure 4.3: Example Grid with Solution

4.3 Pictorial Representation Generation

In this section, I describe in detail the actual system that generates the pictorial representation. In other words, the system that converts the logical form representation into the final picture. I call this the Pictorial Representation Generation (PRG) System.

A top-level description of the pictorial representation generation system is shown, along with close look at each of the components, and *how* the system builds a picture. An example LF representation will be translated into pictorial representation. Each of the modules will play a role in the pictorial generation process.

4.3.1 PRG System Design

The Pictorial Representation Generation (PRG) System, shown in figure 4.4, is the software component of my work. Notice that this figure is basically the same as figure 4.1 without Core Language Engine modules. Each component of the PRG is described in the following sections, along with an example.

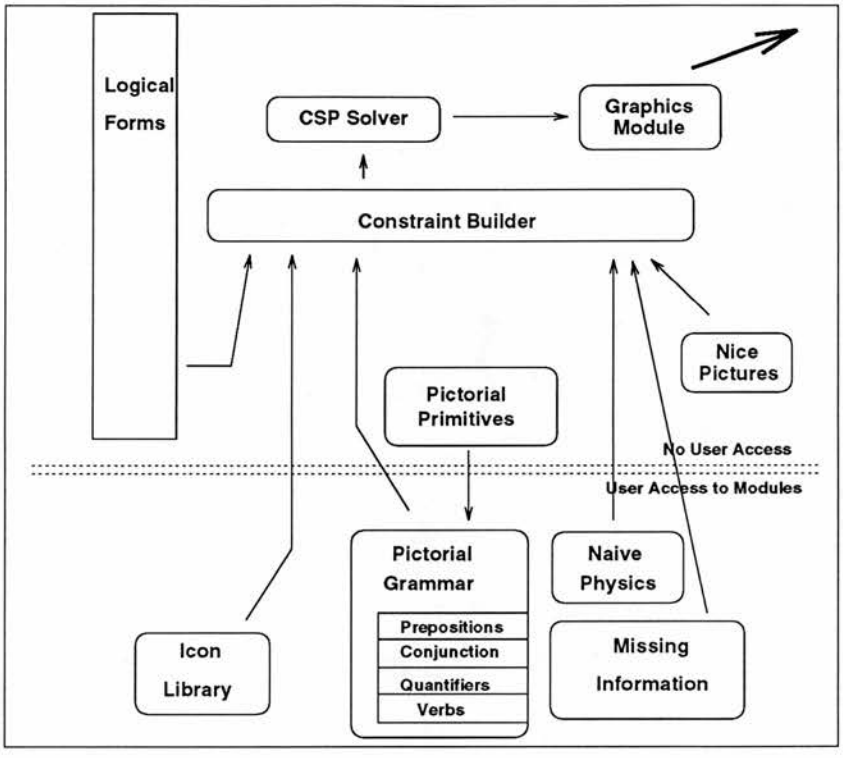


Figure 4.4: Pictorial Representation Generation System

4.3.2 Pictorial Grammar Module

The Pictorial–Grammar (or Sentence–Constraint) Module takes an LF description of a sentence and generates a list of constraints. It is a translation process that converts relationships specified in the LF that act upon the objects in the LF into a set of pictorial constraints acting upon pictorial objects.

Identify the Objects and Relationships

The first step is to identify the objects and relations within a sentence. An example is shown using sentence 15.

(15) The man saw the dog on the hill.

The Logical Form Representation for this sentence is shown in figures 4.5 and 4.6.

```
    %
    % Reading # 1
    % Dog is on the hill AND the man saw the dog
    %

   Complete sentence with bracketing:

     "{the man} saw {{the dog} on {the hill}}."

[dcl,
 quant(exists,
       M,
       [man_MalePerson,M],
       quant(exists,
             D,
             [and,
              [dog_Animal,D],
              quant(exists,
                    H,
                    [hill_HighGround,H],
                    [on_Locational,D,H])],
             quant(exists,
                   E,
                   [and,
                    [event,E],
                    [precedes_in_time,E,SF(date(1991,12,6))]],
                   [see_LookAt,E,M,D])))]
```

Figure 4.5: Example Logical Form, Reading One

Notice that the sentence has two possible interpretations or readings. For this example, we'll only look at the first reading, *Reading One*. *Reading One* has the meaning of "there

```
    %
    % Reading # 2
    % the man saw the dog AND the seeing event took place
    %    on the hill
    %

   Complete sentence with bracketing:

     "{the man} saw {the dog} on {the hill}."

[dcl,
 quant(exists,
       M,
       [man_MalePerson,M],
       quant(exists,
             D,
             [dog_Animal,D],
             quant(exists,
                   E,
                   [and,
                    [event,E],
                    [precedes_in_time,E,SF(date(1991,12,6))]],
                   [and,
                    [see_LookAt,E,M,D],
                    quant(exists,
                          H,
                          [hill_HighGround,H],
                          [on_Locational,E,H])]))))]
```

Figure 4.6: Example Logical Form, Reading Two

is a dog on a hill *and* a man saw that dog."

The relationships are identified by parsing the LF structure[6]. The logical form is made up of several *quant structures.* The third component of each *quant structure* is a *restriction* which makes an assignment of an object to a variable. When parsing each of the restriction clauses, a new object is identified. Another component of the logical form is a *functor* with *arguments.* The *functors* are the relations among the objects.

For example, using the logical form in *Reading One*, the variables assigned to the objects are as: D represents the dog, H represents the hill, M the man, and E represents an event–the seeing event. It is important to note that the 'seeing' *event* is an object, because there is also a 'seeing' relation. The relations are given in a Prolog predicate relating the objects. An example of a relation or predicate is *see_LookAt(E,M,D).* It relates three objects: the seeing event (E), the man (M), and the dog (D). There is another relation described in the example LF–the relation describing the preposition 'on.' It is *on_Locational(D,H).* This relates the dog (D) to the hill (H). In this case, the parsing of the LF representation has identified four objects (E,M,D,H) and two relations[7] (*see. LookAt* and *on_Locational*).

A pictorial object is generated for each object contained in the list of objects found in the LF structure. In the example, the 'on' relation relates the words 'dog' and 'hill.' In the picture, a translated *on* relationship will relate to *pictorial objects.* Each pictorial object has the structure:

```
pictorial_object[Label,IconPtr,X,UX,Y,UY]
```

Where:

- Label is a unique name for each pictorial object. In my system each object is assigned the label of the letters "obj" concatenated with a unique number. An example is "obj1", "obj2", "obj3", and "obj4".
- IconPtr is a tag that is used to retrieve the icon information record from the icon library, which contains the actual graphical image (or icon), and its size information.

[6] For a quick review, the syntax of logical form is described in Section 2.3.3, "CLE Logical Form Representation Scheme." That section also explains how to *read* the LF representation.

[7] I am not counting the 'exists' relationships in the total of the other relationships, as the exists predicate information is captured by looking at the list of objects.

In my system, I use the full disambiguated tag from the CLE lexicon. An example of a tag is "hill_HighGround".

- X and Y are the location of the bottom left corner of the icon in terms of the x and y components of the grid. They can accept integer values.
- UX and UY are the location of the top right corner of the icon (or upper x and y) in terms of the x and y components of the grid. These are needed to represent the size and shape of a pictorial object. They can accept integer values.

When the sentence constraint module identifies an object, the module creates a pictorial object record with just two records filled: the object label, and the tag used to retrieve the icon information record. The location values are left unassigned at this time.

For example, when the LF object is identified from this portion of the LF structure:

```
[hill_HighGround,H],
```

a pictorial object is generated:

```
pictorial_object[obj1,hill_HighGround,X,UX,Y,UY]
```

and has the associated icon information record, which is stored in the Icon Library:

icon[hill. HighGround, SizeX, SizeY, PixelData].

- SizeX and SizeY are the size components (x,y) of the hill icon. For example, the actual values of the hill icon in the working system are 65 and 50 respectively.
- PixelData is a long binary record that is a X-windows icon that is used by Graphics Manager in SICSTUS that represents each pixel of the icon.

After all of the constraints are generated, the size information is taken from the Icon Library, and is used by the constraint satisfaction problem solver to find valid solutions. The next section shows how the constraints are generated.

Generate the Constraints

In the LF representation of sentence 15, two relationships were identified:

- on_Locational(D,H)
- see.LookAt(E,M,D)

The preposition is a relationship among two pictorial objects–D the dog, and H the hill. The preposition *on* is defined in the *preposition sub-module* of the sentence constraint module. The preposition *on* is defined as:

```
on_Locational(A,B) :- above(A,B),
                      touching(A,B).
```

Although the entire pictorial object record is:

```
pictorial_object[obj1,hill_HighGround,X,UX,Y,UY]
```

the following abbreviated form of the pictorial object record is used to show the constraints to improve readability:

```
[hill_HighGround]
```

Applying the 'on' preposition pictorial definition to the dog and the hill, two constraints are generated:

```
above([dog_Animal], [hill_HighGround])
touching([dog_Animal], [hill_HighGround])
```

The verb 'see' generates the following four constraints upon the actor, the verb event, and the object.

left(Actor,Event),
near(Event,Actor),
left(Event,Object),
far-from(Event,Object).

It also gives a command to modify an icon:

highlight(Object)

These constraints and commands basically show the SVO relationship with the actor to the left of the verb, and the verb to the left of the object. Some orientation must be chosen. In the absence of some other information forcing a particular viewpoint, "the actor to the left of the verb, and verb to the left of object" is the default[8]. The constraints generated from the verb also cause the verb icon to be near the actor, and far from the object icon. Finally, the verb constraint highlights the object, to know the exact pictorial object that is object of the relationship. This translates to the following constraints:

```
left([man_MalePerson], [see_LookAt]),
near([see_LookAt], [man_MalePerson]),
left([see_LookAt], [dog_Animal]),
far-from([see_LookAt],[dog_Animal]).
```

In addition to the constraints generated directly from the logical form representation of the sentence, other modules are needed to supply pictorial constraints. The Naive–Physics, Missing–Information, and Nice–Pictures Modules are described in the following sections.

4.3.3 Naive–Physics–Constraints Module

This module supplies a small set of constraints for naive physics. Previously, the need for gravity was shown in "the spider on the wall" example. The gravity constraint is applied to all objects, unless the gravity feature is off for an object. There are a few

[8] Further discussion of the SVO relationship is given in section 5.6.

objects, such as birds or a spider hanging on the wall, that can defy gravity. Without the gravity constraint the possible solutions to placement of the objects can be very large–too many legal solutions. Another reason for using the naive–physics constraints, is that may simplify the definition (reduction in number of constraints) of many of spatial prepositions.

If Object A does not contain the feature [gravity -] listed in its icon information record[9], then the gravity rule applies. The rule basically says that every object must touch the ground unless it is already resting upon some other object[10]. The ground exists as an object and is not affected by gravity. Also, the ground is always at the bottom of the grid.

For each object in object list:

```
touching(A,ground).
```

This constraint is applied with a lower priority than the other constraints generated directly from the LF structure. In this way each object wants to touch the ground, unless it is constrained by other constraints with higher priority. The naive–physics constraints are added to the total list of constraints.

4.3.4 Missing–Information–Constraints Module

The Missing–Information–Constraints Module has two goals. It provides constraints to apply in the absence of specific information. To draw a picture, specific information is required. Second, it attempts to reduce false implicatures–in other words, when the viewer interprets something from the picture that was not intended in the sentence. Look at figure 4.7.

Where does one draw the cup? Additional information is needed to provide an exact placement of the cup. The cup may be placed on the left edge, or the right edge of the

[9] Although this information may not seem appropriate to be stored in the icon library, it must be stored there or in the lexicon. It was decided that the features related to text should be stored in the lexicon, and the features for the pictorial representation to be stored in the icon library.

[10] I am not worrying in this example if an object can really support another object. Although not implemented in my system, that could be another naive–physics constraint, if so desired.

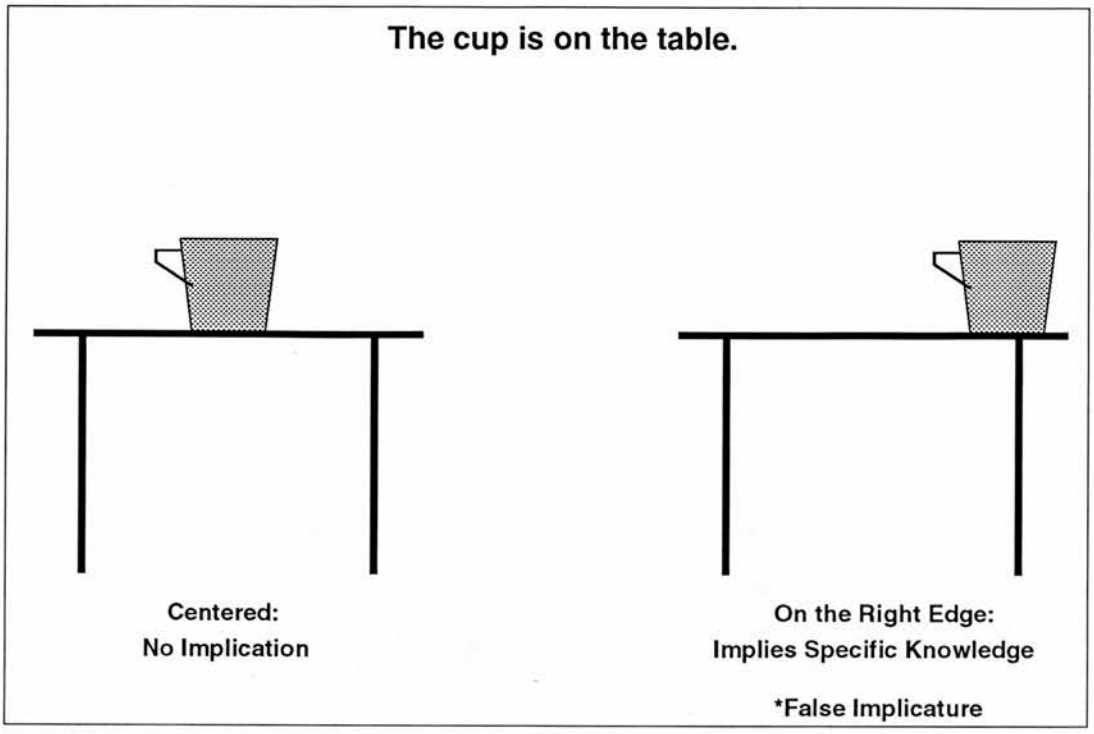


Figure 4.7: Missing Information Constraint Example

table. However, this seems to imply other information (i.e. greater specificity) that was not intended. The viewer may think that the reason the cup is on the right in this picture is because the sentence contained information stating that the cup is really on the right side of the table. If a picture shows the cup is in the center, it seems to imply the cup could be anywhere on the table including in the center (i.e. less specificity).

The implementation restricts positions that tend to produce false implicatures. A constraint of "when two objects are touching, the smaller object is neither on the left side nor the right side of the bigger object" (in the absence of other information stating that it must in a particular position), prevents the false implicature in the above example. This constraint is given a very low priority, so that if some other information does provide a specific position, then the missing-information constraint is relaxed.

4.3.5 Nice–Pictures–Constraints Module

The Nice–Pictures–Constraints Module provides some additional constraints. Most of the constraints do not provide any significant change to the picture except to make the

picture more pleasing to look at. This includes centering the picture and keeping a margin at the edges of the grid.

However, one significant constraint is that icons cannot overlap. When drawing the picture for the first time for sentence (16), I saw the icon for a man, the icon representing the seeing event, and the cat. I did not see a icon for the dog. What had happened was that a solution was found placing the dog and cat in the same location in the picture. When this constraint was added, the system showed all of the icons in their respective positions.

(16) The man saw the dog with the cat.

4.3.6 Constraint Builder

The Constraint Builder is the controller of the PRG System. It takes logical form description and asks the Pictorial Grammar Module to identify the objects, and the constraints among these objects. It then asks the other modules to apply constraints. After all of the constraint modules have operated, the constraint builder has a large list of constraints plus a short list of the objects that exist. The constraint structure for this example is shown in figure 4.8.

This controlling module also handles the highlighting of the object icon. This is done by a modify–icon command. When the constraint list is generated, the highlight(object) is also generated. This involves setting a highlight flag in the pictorial object record. When the icon record is being retrieved for the associated pictorial object record, it checks if the highlight flag is set. If so, a highlighted version of the icon is used.

The final stage of the constraint builder is to order the constraints by their priorities. The constraints with highest priority are listed first, with the lowest priority constraint listed last. The constraint structure and a list of all of the objects is then sent to the Constraint Satisfaction Problem Solver.

4.3.7 CSP Solver

After all of the constraints are added to the constraint list, the list is normalized by the constraint builder. Then, the ordered list of constraints along with a list of the objects is

```
THE SENTENCE:
     "{the man} saw {{the dog} on {the hill}}."

THE CONSTRAINT STRUCTURE:

    left([man_MalePerson], [see_LookAt]),
    near([see_LookAt], [man_MalePerson]),
    left([see_LookAt], [dog_Animal]),
    far-from([see_LookAt], [dog_Animal]),
    above([dog_Animal], [hill_HighGround]),
    touching([dog_Animal], [hill_HighGround]),

    touching([man_MalePerson],[ground]),
    touching([see_LookAt],[ground]),
    touching([dog_Animal],[ground]),
    touching([hill_HighGround],[ground]),

    not left([dog_Animal],[hill_HighGround]),
    not right([dog_Animal], [hill_HighGround]),

    not overlap([man_MalePerson], [see_LookAt]),
    not overlap([man_MalePerson], [dog_Animal]),
    not overlap([man_MalePerson], [hill_HighGround]),
    not overlap([see_LookAt], [dog_Animal]),
    not overlap([man_MalePerson], [hill_HighGround]),
    not overlap([dog_Animal], [hill_HighGround]).
```

Figure 4.8: Total List of Constraints for Example

given to the Constraint Satisfaction Problem Solver. The only other piece of information that is needed is the workspace information—the imaginary coordinate space. In the example for "the cat on the table," the imaginary coordinate space was a five by four grid. Each icon was assigned to one of these points. In the second example, icons of varying size were used, and the use of a much larger grid. However, the problem is still reduced to a set of constraints to be solved, by legal placement of the icons.

A simple depth-first backtracking method was used in the cat on the table example. This method is not efficient and becomes worse as the grid size increases. The worst possible search, or a search that looks for all solutions is:

$$\psi = \frac{\omega(G_x * G_y)^n}{2}$$

Where:

ψ is the worst case search (in number of constraint checks)
ω is the number of constraints
G_x is the size of the grid in the X axis
G_y is the size of the grid in the Y axis
n is the number of objects

There are several techniques to improve the efficiency of the search. *Intelligent* backtracking techniques could be used, such as Dependency Directed Backtracking, or forward checking procedures. In more sophisticated systems, Justification Truth Maintenance Systems and Assumption Truth Maintenance Systems are also used [Doyle 79], [deKleer 86a], [deKleer 86b]. This was not chosen because of the time required to build a truth maintenance system for this project.

In the first version of this system, the software ran on an IBM Personal Computer with VGA screen. The time to solve even a very small grid of size of five by five with twenty constraints took several hours for each picture to be generated. Therefore, I implemented a set of heuristics to place the objects. I placed each object by moving it as far as possible in the direction that the constraint intends. If a constraint is above(A,B), I would move icon A well above B, by using a *bump* value. Later other constraints will move the

icon back to within their limits. After bumping one icon, I would then re–check the constraints to see if they are still satisfied, with special consideration of checking objects already placed. The heuristics may have to be applied to some of the objects already placed. If the set of heuristics could not find a solution, an alternate set of heuristics was applied. This approach yielded a possible solution for each list of constraints, and more importantly was computationally inexpensive.

Eventually, the set of heuristics was not practical if the set of constraints became large. It also would fail on about 3% of the cases, leaving only the expensive depth-first search to yield a solution for those cases. A third problem was that it would only produce one solution rather than give the range of legal solutions. Therefore, in the final system used in this thesis, a more sophisticated constraint solving approach was adopted.

One interesting technique is Hierarchal Constraint Logic Programming, as described in [Borning *et al* 89], [Wilson 89], [Freeman-Benson & Wilson 89]. Another CSP technique is to solve the problem by handling the constraints directly, and then applying the solution to a specific grid. In this way the grid size does not effect the solving of the CSP. In this situation, the problem is *solved* and only applied to a grid at the end of the problem.

CHIP (Constraint Handling In Prolog) is an example of a system that can do that [Dincbas *et al* 88a], [Dincbas *et al* 88b], [Dincbas *et al* 88c], and was used in the thesis system to solve the constraint satisfaction problem. A notable advantage was that the *ranges* of legal solutions were given. This allowed for constraints to be applied in an interleaved fashion. For example, all of the constraints generated from the logical form are given to the CHIP system. CHIP yields the range of solutions for each of the objects. Then the naive-physics constraints are applied. Finally, missing-information constraints can be applied if they are needed. If an object was already forced into a certain position, it would not need further constraints from the missing-information module. However, if the range of legal solutions was large, then the missing information module would constrain the solution.

4.3.8 Graphics Module

The graphics module consists of two process. First the placement of pictorial representation windows, and second, the drawing the icons within the window. The placement

of windows, and the drawing of the icons are done via SICSTUS GM (Graphics Manager) commands which interface to X-windows on a SUN work station.

The placement of the windows is fixed in the current system. Most representations involve just one spatial Pictorial Representation Window (PRW) and one temporal PRW. The temporal PRW is attached to the bottom of the spatial PRW, as the two windows are pictorial representing the same event.

When the special case of multiple spatial PRWs is needed, they are placed in a predetermined way. Perhaps in a future system, the placement of windows could be handled via a set of constraints. However, it was not seen to be required for this project.

The graphics module displays all of the pictorial representations for a sentence at the same time. One icon definition window is given to tell the meaning of the icons that are used. The icons are defined by the user using the X-windows command **bitmap**. Icons can be of any size, and can be in reverse video as well.

A future improvement would be to use a three-dimensional graphics module. If the objects were represented as three-dimensional objects, the system may be able to use some software that could take a set of points describing the objects and a given viewpoint and draw the appropriate picture. Since the system is designed modularly, the PRG system should be able to link up with an available graphics package. Three dimesional graphics can depict more spatial relations than two-dimensional graphics, and also can make use of other constraints. Three-dimensional constraints are discussed in section 6.1.2.

4.4 Evaluation

The output of the Text-To-Pictures System is shown in the next few chapters covering several different linguistic types of text. These pictorial representations evolved, several times, from user feedback.

The output of the system was shown to approximately twenty people, who attempted to interpret a pictorial representation without any previous description or instructions. In the beginning, some pictorial representations were ambiguous or implied other information not intended.

Although there were many changes and improvements, a few are worth mentioning. Originally, gravity was not implemented in the system. For some pictures, a "legal" solution would show an object above the ground. Often, the subjects could understand the gist of the picture, but the lack of gravity was distracting, primarily because it was random(some cases it appeared to have gravity in effect, and in others not, because both were legal solutions). Adding the gravity constraint described in section 6.5 solved that problem. In that section, an example of pictorial representation, without the gravity constraint implemented is shown.

Other feedback involved the problem of distinguishing between the accompaniment case or the instrument case (in an example involving a telescope which is shown in detail in chapter 8). Examples of the accompaniment and instrument case are also shown in the next chapter. Several positions of placing the telescope next to the actor were tried. The only one that did not generate a lot of confusion, was to place the instrument between the actor and the object, and to place the telescope to the outside (not between) og the actor or object for the accompaniment case. The modification of the icon, to show it being used as an instrument, also helped distinguish it from the accompaniment case.

(17) Carla saw the man on the hill with a telescope.

An interesting case evolved because the Core Language Engine's logical form shows the preposition modifying the event rather than the actor. This lead to a problem in the pictorial representation. For example, in sentence 17, the man could be on the hill, or the seeing event took place on the hill. The logical form representation of the "on" preposition generated a constraint to locate the seeing event *on* the hill. This would mean that icon to represent *see* would be located on the center of the hill, where Carla would be located off-center. This was confusing to viewer, who again, could get the gist of the sentence, but wondered why the actor was not centered on the hill. This was solved by having the constraints that were to operate upon the location of the event, to actually operate upon the actor. See the example in figure 4.9, where the first representation shows the event "on" the hill, rather than the actor, and the second representation shows the actor "on" the hill.

Interestingly, the subjective definitions of the pictorial primitives *near*, and *far-from* were

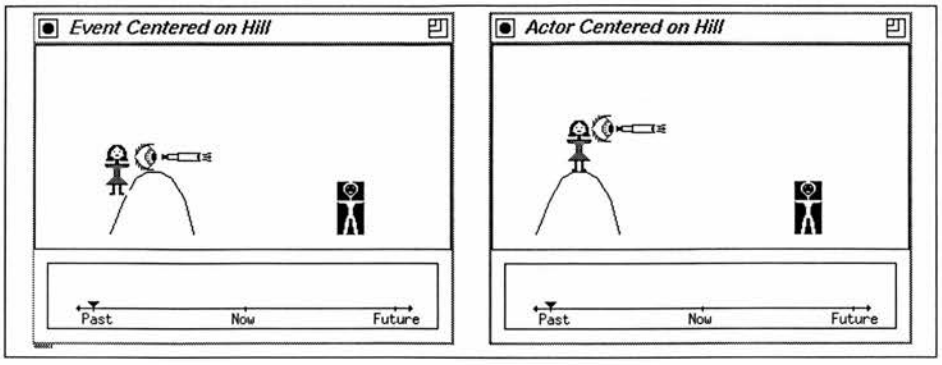


Figure 4.9: Preposition Operating upon Event, and then upon Actor

not critical. This was true if *near* was not so close to be confused with *touching*. Several subjective distances were chosen, with little change in understanding by the viewers.

Approximately 70% of the questions from the users concerned the meaning of certain icons. Eventually, an icon definition window was added, which allowed an untrained user to quickly determine the intended meaning of an icon. The *icon definition window* is shown at the beginning of the next chapter.

The use of subject feedback was an informal study, and its sole intent was to improve the system rather than determine human perception characteristics.

4.5 Summary

This chapter shows the techniques used in, and the design of, a small text-to-picture system. The system is based upon the concept that an object can be represented by an icon. Therefore, the problem is reduced to defining pictorial representation windows (PRWs), finding the appropriate placement of the icons within the PRW, and modification to icons, to provide a pictorial representation. A constraint-based approach seems to work well, and it is interesting that the system works with a very small set of pictorial primitives, which are fully described in Chapter 6, "Spatial Expressions" and Chapter 7, "Temporal Expressions."

This system could be used by a wide group of people. One group could be researchers in spatial expressions, naive physics, knowledge representation, et cetera. The system

could also be used as a tool to display all the possible meanings of sentence when it is ambiguous and to let the viewer decide the one intended by the speaker. A pictorial representation of the sentence is easier to decipher than a logical form or parse tree, especially for a user without a linguistic background. This ambiguity application is implemented and described in Chapter 8, "Ambiguity and Vagueness." Finally, the system may be applicable as a tool in second language acquisition.

The system incorporates a modular design. The domain–dependent modules were separated from the rest of the system. In this way, researchers who want to study spatial expressions, or naive physics, or some other area can change their own module, while retaining the rest of the system.

I used this system to identify and describe the inter–relationships between text and pictures by experimentation [Ludlow 89]. Several types of linguistic expressions have been examined, with the associated translation technique, and a generate pictorial representation. They are described in the following chapters five through eight, covering noun phrases, number, scoping, conjunction, relative clauses, some verb features, spatial expressions, and temporal expressions.

Chapter 5

General Expressions

There are several types of linguistic expressions that can be represented pictorially. This chapter will present a sample of the linguistic types that can be represented. The thesis concept makes use of the idea that nouns or objects can be represented by icons. A discussion of the concept of icon representation and potential problems is given. The icon definition window, and computation requirements are also discussed. Other features of noun phrases such as number and relative clauses are pictorially represented. Conjunction is handled by use of pictorial representation windows. An example of collective versus distributive scoping is demonstrated. The SVO verb structure is captured and negation is demonstrated. This is a large collection of several general linguistic expressions with the exception of spatial or temporal expressions which are covered in the next two chapters.

5.1 Object Concept

Herskovits discusses that there have traditionally been two ways to conceive of the meaning(s) of a lexical item: a list of conditions or the notion of a prototype [Herskovits 86]. The list of conditions can be a set of a truth-conditions contributing to the sentence meaning. This is sometimes called the "checklist" approach [Fillmore 75]. The **prototype method** of representing meaning for a lexical item is the concept that is used to pictorially represent an object.

Several *prototypes* of natural kinds were studied from a psychological perspective described in [Jung 64], [Rosch 77]. Herskovits discusses how a bird is represented by "best instance" of a bird. Among North Americans, they seemed to have a quite similar de-

scription of a prototypical bird (in terms of size, shape, color, et cetera), which was somewhat like a robin. Often a prototypical image can represent a class of objects. This technique should be kept in mind by the user when constructing an icon to represent a set or class of objects.

Chapter three looked at the reverse of the text–to–pictures system. In generating text to describe a picture, it becomes very important in simply choosing a word to textually represent a visual object. Jerry Fodor describes the problem of looking out a window and seeing "a lady walking a dog." He states that the viewer could have also described the situation as "a lady walking an animal" or even "a lady walking a silver-gray, miniature, poodle bitch" [Fodor 83, page 96]. He gives a *cluster of psychological properties* that tend to describe a preference that a viewer would use in textually describing a visual scene.

The selection of the where to locate objects within a pictorial representation window to pictorially representation relations among the objects is the responsibility of the system. However, the icons to represent the objects must be defined by the user. The system represents Fodor's example, by using a user–defined icon of a prototypical pictorial representation of a dog, that is a different icon from 'animal' or 'poodle', just as the speaker chose the word 'dog' rather than 'animal' or 'poodle'. If the word 'poodle' was used specifically, then, then a 'poodle' icon should be used.

The prototypical definitions of objects are defined by the user. This feature makes the system much more manageable (and useful). It is often difficult for the user to develop an icon to represent a 'dog', that could not be confused for one of 'poodle,' 'alsatian,' or 'German shepherd'. A good technique would be to not allow specific nouns to have same icon as generic nouns. The icon dog should not look like a specific dog. A unique icon should be used to represent a generic object. Some references do exist which try to provide a common definitions of pictorial representations of objects such a pictorial dictionaries [Dunden 89] and [Parnwell 91].

The actual pictorial definitions of objects seem to be rather domain specific and user specific. Therefore, the icons are stored into an icon library which is in separate module from the rest of the system. The user is able to define the icons by using the *bitmap* command the generates icons for X-windows.

5.2 Computational Requirements

Objects each have a object record that contains the edge and size information. The record for objects A and B is:

pictorial_object[ObjID. A, Tag_A, Left_A, Right_A, Bottom. A, Top_A, Front_A, Back_A]
pictorial_object[ObjID. B, Tag_B, Left_B, Right_B, Bottom_B, Top_B, Front_B, Back_B]

Where:

- *Obj_ID* is the Object ID Number
- *Tag* is the actual CLE lexicon tag for a word (e.g. table_Furniture)
- *Left, Right, Top* and *Bottom* are the edges of the icon. The size can be determined by subtracting the *Bottom* from the *Top*, et cetera.
- *Front* and *Back* are similar to *Left* and *Right*, but are only used for a three-dimensional system.

When one object is constrained compared to another object, the primitives act upon the edges of the icon. In this way, the size information of an icon is taken into account. In using size, a uniform scaling of the actual size would be impossible. However, the relative size information should still be intact. A larger item should appear to be larger (even if not uniformly larger). Also a limit is implemented that extremely large or minuscule objects can still be placed within a window or seen by the viewer.

The size of icon was chosen by a guideline that the man icon was set to 40 pixels and assume a man is 6 feet tall. The other objects were chosen using the logarithmic scale in comparison to the man icon. The following example determines the size of a cat icon, and of a hill. If a cat is guessed to be 18 inches (1.5 feet) high, and a hill to be 600 feet high, then the ratios are determined as:

$$\frac{Cat\ Size}{Man\ Size} \equiv \frac{1.5}{6.0} = 0.25$$
$$\frac{Hill\ Size}{Man\ Size} \equiv \frac{60}{6.0} = 100$$

The logarithms of the ratios are taken and then pixel size is determined:

$$\log Cat\ Ratio \equiv \log 0.25 = -0.602$$
$$\log Hill\ Ratio \equiv \log 100 = +2.00$$

The size is calculated by multiplying the (log value + 1) times the standard value (man at 40 pixels):

$$\log(Cat\ Ratio) + 1 = 0.398, \quad 0.398 * 40 \approx 16\ pixels$$
$$\log(Hill\ Ratio) + 1 = 3.00, \quad 3.00 * 40 = 120\ pixels$$

Therefore, the 'Cat' icon should be 16 pixels high, and the 'Hill' icon 120 pixels high. Assuming a window of size 300 pixels (horizontal) by 200 pixels (vertical), the objects of less than 8 pixels will be set to 8 pixels in height, and objects greater than 200 pixels will be maxed at 200. This technique seems to preserve the relative difference in size, but allows the icons to be visible and also fit within the pictorial representation window.

5.3 Noun Phrases

Noun Phrases are made up of nouns (or objects), adjectival modifiers, determiners or quantifiers, relative clauses, and prepositional phrases, and clauses [Mellish 85]. This section will show how a pictorial representation is generated for most of these components of a noun phrase.

Once an icon exists to represent the object of the noun phrase, the possible modifications to the object need to be represented. These modifications include number, relative clauses, adjectival descriptions, and prepositional phrases. Number and relative clauses can be handled in a straight forward approach as will be shown. Adjectives are much more difficult to represent, as different *classes of adjectives* are represented differently. Most prepositional phrases are considered to be of two types: spatial and temporal. These type of prepositional phrases are not discussed here because they warrant their own chapter. Spatial expressions are presented in the next chapter, and temporal expressions in Chapter 7. However, some other types of prepositions (i.e. ones that show instrument and accompaniment) are discussed in the last section of 5.3, "Noun Phrases."

5.3.1 Number

Number is displayed in two different ways depending upon the number. If the number of objects is one or two, then that many actual objects are shown. When the number is three or more, a different icon representation is used to represent 'many' objects. This is motivated from some number systems that are "one, two, and many" and the Hindu language that has noun-verb agreement of one, two, and many [Kielhorn 12],[Burrow 55].

The single icon without a number represents the cases when the number was determined by 'one', 'the', 'an', or 'a.' Two icons represents 'two.' The icon representation of *many* is a single icon that shows three of the objects close together. The number of the *many* is also superimposed over the icon. For example, in the following expression, the number of policemen is represented with two icons of a policeman. In the other expression, the number of criminals is shown by a single icon of 'many' criminals, and the number '7' superimposed over the icon of criminals.

(18) A policeman saw the shooting.

(19) Two policemen drove to the football stadium.

(20) Seven policemen arrested hooligans.

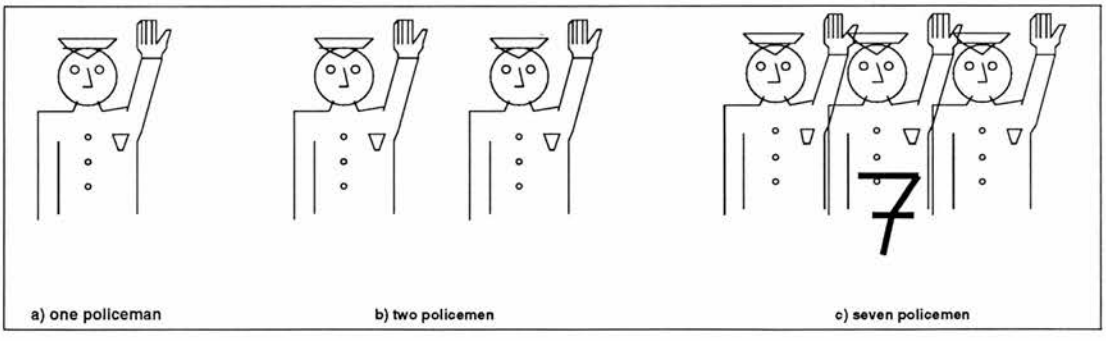


Figure 5.1: PR of Number

The method of choosing the *many* icon is considered as an icon modification via a 'replacement' of the *single-number icon* with the *many-number icon.* Then an icon of the actual number is placed into the system, with the constraint 'overlap(n, i),' where n is the *number icon*, and i represents the *many-number icon* of the object.

The technique of representing 'two' is more difficult than representing 'many', because another pictorial object is introduced. The constraint of near(A,B) is introduced, where A is the icon for the object, and B has the same icon representation as A. However, there are two objects. This also involves introducing another object into the visual object list. The 'near(A,B)' constraint does not require that the two objects touch. They may or not touch depending upon the other constraints generated by the rest of the sentence.

Some may argue that other numbers, other than 'one' or 'two', should be represented by showing the actual number of objects. However, this definition shows the capability of the system, in that two methods of representation are used. Choosing a different technique of when to apply the many–icon–method or when to actually display the 'more–than–one' icon, will not add any additional representation mechanism. In other words, if 'three' were to be represented in this fashion, it would follow the same pattern as how 'two' is represented.

An example of when a separate icon is needed to represent more than the number of two, is in discourse. Given the following text, the representation would need to have a separate icon for each man.

(21) Three men robbed the store.

(22) The first man had a gun and stood at the door.

(23) The second man took the money, while the third man waited in the getaway car.

A further discussion of using a distinct icon for each object within a same class to identify it as being unique is seen in the relative clause and embedded sentence examples.

Another situation of *number*, is the special case of *none* and *no one*. In this case, an object does not exist, however, a pictorial representation must show the object so the viewer knows of what type the object was that does not exist.

An example is to say that *no one* read the book. The technique chosen is to use an icon to represent 'none', 'no one', or 'zero'. The icon representation is to use the icon of one of the objects and superimpose the number '0' over it. This was similar to the technique of representing number of many.

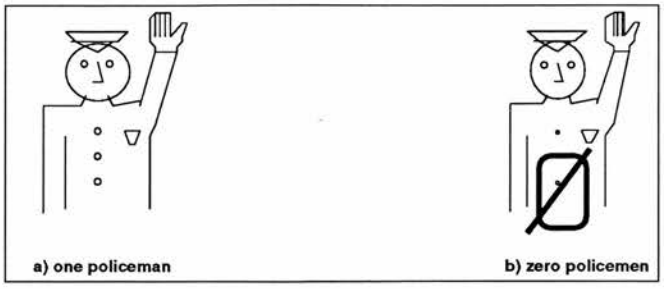


Figure 5.2: PR of *none*

A distinction between definite and indefinite quantifiers was not made in the pictorial representation. This was primarily because there was no distinction needed in the representation for handling single sentences that did not involve discourse. In future work, some pictorial distinction between definite and indefinite quantifiers should be made. In addition, universal quantifiers were not handled within the thesis system. McCord [McCord 82] offers some definitions of universal quantifiers in terms of percentages. For example, "most" would mean more than fifty percent. To display these universal quantifiers, some additional technique would be needed. It would probably involved handling complex icons, or using large numbers of icons. In both cases the approach would be clumsy, and it could be unwieldy in developing these complex icons. For these reasons, universal quantifiers were not attempted in this system.

5.3.2 Relative Clauses

Relative clauses modify the object. The modification is represented by another pictorial representation window (PRW) that modifies the object. The following sentences are examples[1].

(24) The man saw the cat on the table.

(25) The man who drove the car saw the cat on the table.

In sentence 24, the 'man' is shown in a visual scene as "sees the cat." This is represented in a pictorial representation window. In sentence 25, the 'man' is modified by "who drove

[1] This example was previously seen in Chapter 3.

the car." In figure 5.3, this is shown by the PRW1 showing the 'man' and the main action of the scene and another PRW, labeled PRW3, showing the modification to the 'man', who is "driving the car."

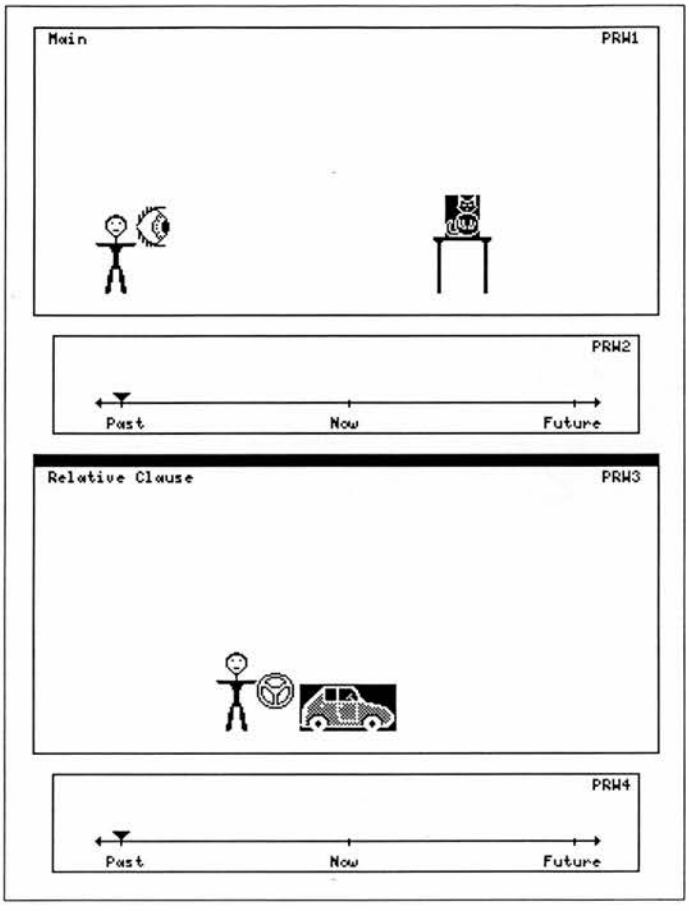


Figure 5.3: PR of Relative Clause

This same technique is used to represent embedded sentences. Since the pictorial representation is a recursive definition, it can handle embedded sentences without modification. In the sentence 26, the top-level event is that "Alan said that X." X in this cases represents what Alan said, and is also another event. This second event is represented with another pictorial representation window showing the that event, which is "Ian saw the woman on Tuesday."

(26) Alan said Ian saw the woman on Tuesday.

The pictorial representation for sentence 26 is shown in figure 5.4.

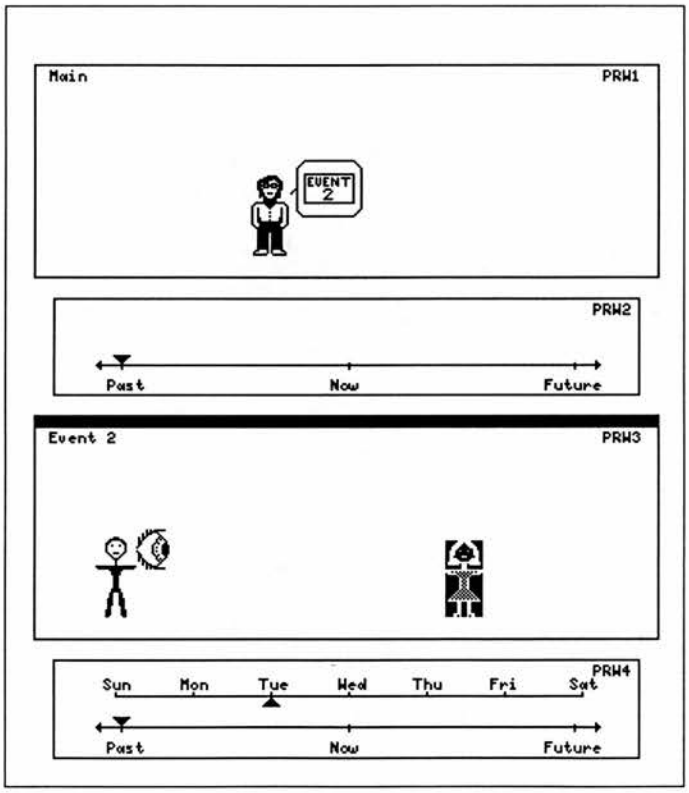


Figure 5.4: PR of Embedded Sentence

The second group of pictorial representation windows (PRW3 and PRW4) represent the subordination of another sentence or event. In the previous example in figure 5.3, relative clauses were shown, which is the subordination of information about an object. The only distinction between the two pictorial representations are a textual marking in the upper left corner of the window which states "relative clause" or "event 2." This violates the pictorial representation method of using icons, spatially locating icons, or modifying icons as the only means of representation. The reason for this deviation was because the graphical software did not allow an implementation to specifically show *what* (the object or an event) is being subordinately represented in the second group of windows.

Given a different graphic software package, an exploding window technique could be used. For the relative clause, the "man" icon, when clicked on by the mouse, could explode into the second group of PRWs that represent "who drove the car." In the embedded sentence example, the bubble showing an event could be clicked on to explode into the second group of PRWs that show that "Ian saw the woman on Tuesday." In this way, the representation would use spatial relations rather than textual stating a grammatical relation.

A non-liner representation method, such as Hypertext, could have been used to handle the embedded pictorial information associated to an object in the top level. The hypertext technique was not chosen, because the power of its representation cannot be shown on one page. It would be extremely difficult to illustrate the full pictorial representation of a sentence using a non-linear representation within the pages of a thesis report. The method of representation shown in this thesis could always be converted into a non-linear representation, however the converse is not true.

The thesis system does not handle discourse, however, several objects in one sentence could be introduced that could interfere with each other in terms of interpretation. In these cases it is best to have a unique icon representing one of the men, and another unique icon to represent another man. In the logical form each man is identified separately as *man1*, *man2*, et cetera.

The previous example had two different 'man' icons–one for Alan, and one for Ian. However, given sentence 27, 'he' is ambiguous as either Alan or some other man. The pictorial representation shows both of the interpretations. In the first picture, the Alan icon is

used to represent 'he'. In the second picture, a generic man icon is used to represent 'he'.

(27) Alan said he saw the woman on Tuesday.

The pictorial representation showing same person in two PRW (Alan doing the saying event, and the seeing event) is represented in figure 5.5.

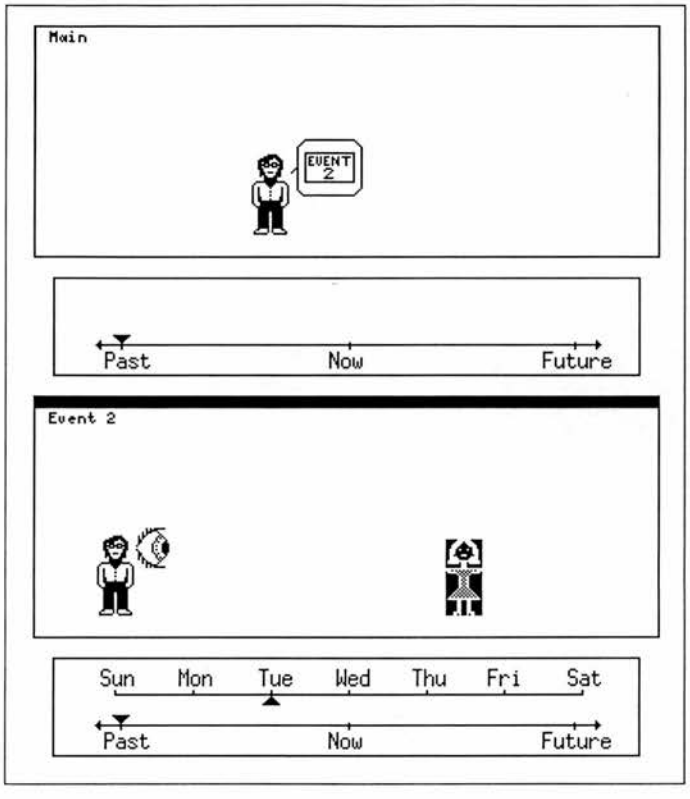


Figure 5.5: PR of *same* Actor in Two Events

If the exploding icon method, or some other technique that clearly associates a subordinate PRW to a specific object, is used, then the technique of using two different icons to represent two different men is no longer required.

5.3.3 Adjectives

Adjectives are not handled by the Text-To-Pictures System in this thesis. There does not appear to be some generalized method of handling adjectives, therefore they were not implemented into this system. However, with in a limited subject domain, one can

imagine making use of several adjective classes, such as color, size, texture, temperature, et cetera.

Color was not used in the implemented system for two reasons. The SUN work stations that I used were not color. Second, I did not want to use color to distinguish or represent relational information, because if later color was implemented it would conflict with other meanings in the pictorial representation, such as the color of an object. It should be relatively easy to implement a mapping of adjective color descriptions to a color in the pictorial representation. Color could also serve to make icons unique, as in sentence 28.

(28) He stood next to the blue car.

The picture could have more than one car, but with only one of them blue, which is the one where the man icon should be located next to.

Other adjective descriptors such as size, are not as easy to pictorially represent. Given the following sentence 29, it would be unclear how to draw a small elephant, unless there were several different icons to represent each different adjective modification of an elephant. A more principled way would be some technique of modifying the icons in a general way.

(29) The big mouse scared the small elephant.

5.3.4 Prepositional Phrases

There are several types of prepositional phrases that could represent instrument, accompaniment, benefactor, locative, or temporal information [Fillmore 68]. Locative and temporal prepositions are discussed in the next two chapters, while the instrument and accompaniment prepositions are presented here.

In sentence 30, a'telescope' is *accompanying* 'Alan'. In sentence 31, the 'telescope' is the *instrument* that 'Ken' used to see Alan.

(30) {{Ken} saw {{Alan} with {a telescope}}}.

(31) {{Ken} saw {Alan} with {a telescope}}.

Accompaniment is easier to represent than instrument. The definition of the *with* preposition with the *with_Accompanying* sense is defined as:

touching(B,A),
not(above(B,A)),
not(below(B,A)),
not(between,B,Actor,Object))

These constraints force the telescope to touch Alan, but not be above or below him. This forces the telescope to be to left or right side of Alan. Finally, the telescope is put away from the center region between the actor and the object (Alan). This is so that it could not be confused as having any current use as an instrument.

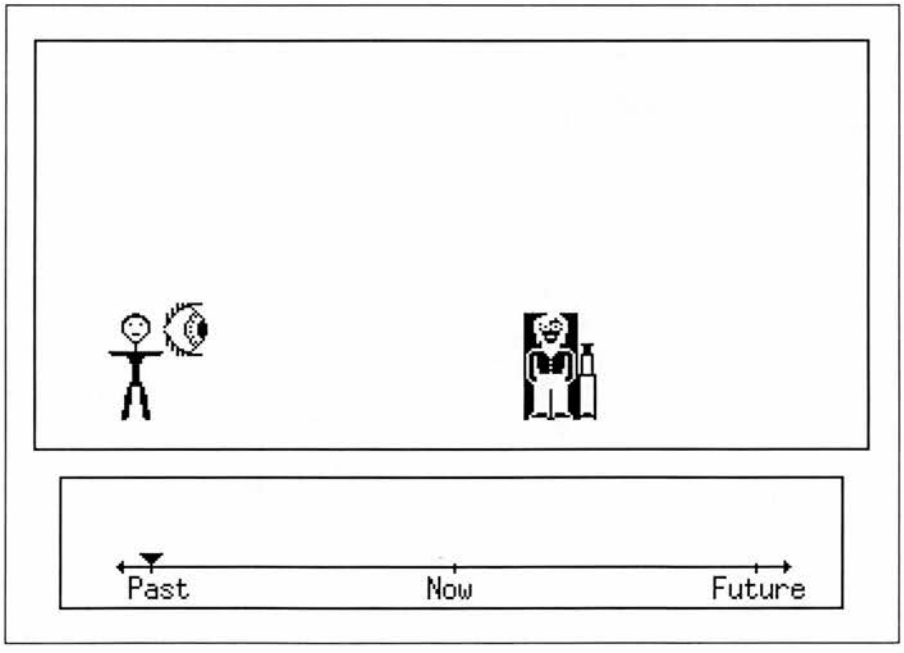


Figure 5.6: PR of Accompaniment

In the instrument case, the actor (ken), the event icon (seeing), and the instrument icon (telescope) are needed. The instrument icon is constrained to touch the event icon to show usage. It is also placed in between the actor and the patient of the sentence, so that it shows the instrument as having some involvement with the event.

The *with* preposition definition for instrument is defined as:

between(Instrument,Event_Icon,Patient),

touching(Instrument,Event_Icon),

in_middle_vert(Event_Icon,Instrument)

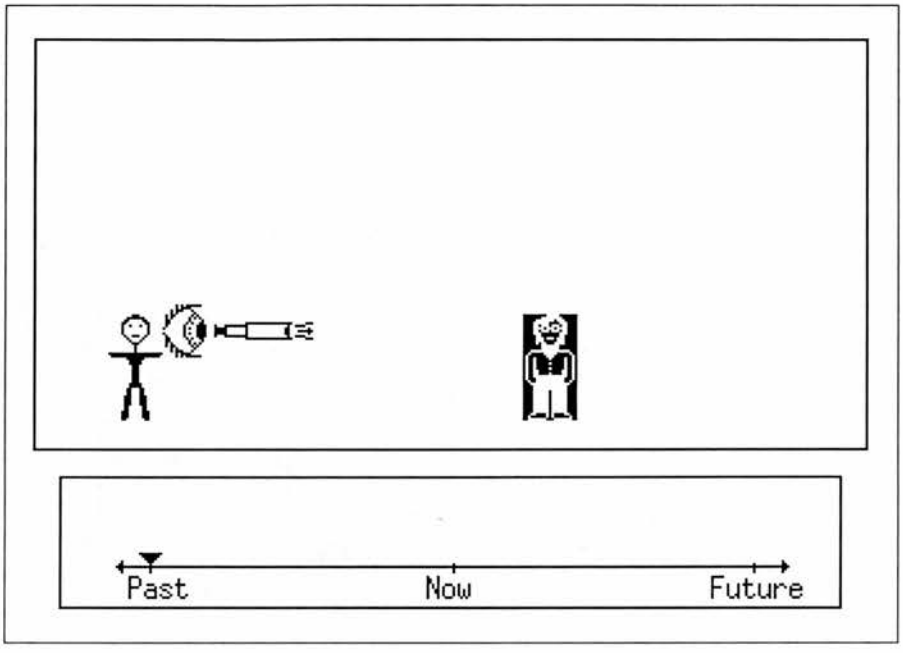


Figure 5.7: PR of Instrument

The instrument is identified because it is known to be a tool of the activity portrayed. For example in figure 5.7, if the 'telescope' is shown 'near' 'Ken' in the activity of seeing, it will be deduced that the 'telescope' is the instrument and was not accompanying Ken.

5.4 Conjunction

Conjunction is pictorially represented by a exploiting how the pictorial representation windows are used. The *and* and *or* conjunction are presented. The 'not' function is described at the end of this chapter in section 5.7, "Negation."

Conjunction presents many of the same problems to a pictorial representation as discourse presents. Given a *S-and-S* (Sentence and Sentence) construction, one could rewrite the sentence as two sentences and look at the text as a discourse problem. However, the conjunction can occur between two NPs or two VPs or maybe between two NPs in a PP. These will all have different consequences on the pictorial representation.

The pictorial technique of handling conjunction will use two of the three methods of the pictorial representation concept described in Chapter 3. The primary technique will be the use of another pictorial representation window to represent the other event. Additionally, placement of another icon within an existing pictorial representation window may be required.

There are several types of *and* conjunction. The following are examples taken from Rob Milne's thesis [Milne 83]. His ROBBIE system could handle:

(32) The boy and the girl hit Mary.

(33) The boy hit the girl in the park and in the head.

(34) The boy hit and kissed the girl.

(35) The boy hit the girl in the park and street.

(36) The boy hit Sue and kissed Mary.

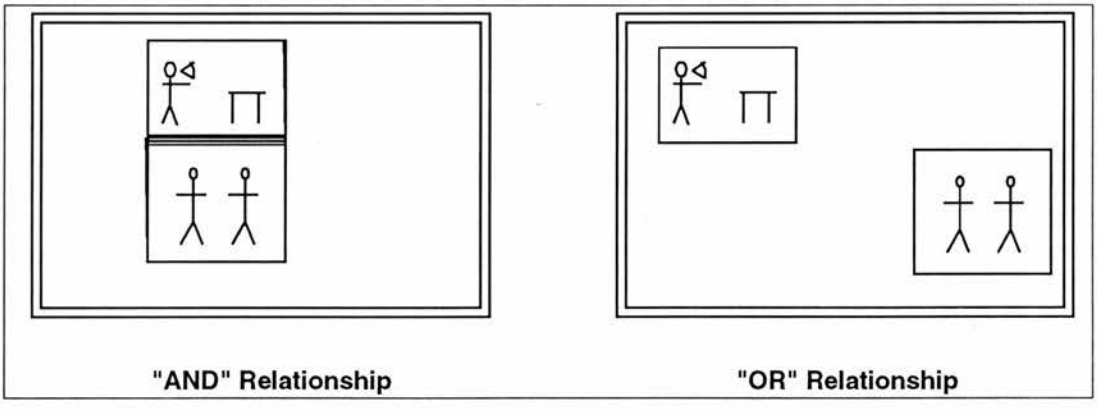


Figure 5.8: "*And* and *Or*" PRWs

The *and* relation involves splitting the one sentence into two events. In sentence 32, there are two events. The first is that the boy hit Mary. The second is the girl hit Mary. In sentence 34, the two events are the boy hit Mary, and another event of the boy kissed Mary. To show that these two events are related by the *and* relation, the two pictorial representation windows are joined together as is seen in figure 5.8. The *or* relation is that the two pictorial representation windows are used each showing an event, but they are *not* touching, and act as unrelated to each other.

For example, starting at the top level of a sentence, a PRW is created with the marking "Pictorial Representation #1." However, if given an ambiguous sentence (it is either one pictorial representation *or* another representation), then two PRWs would be created at the top level. This new PRW would be marked "Pictorial Representation #2." An example of that is shown in figure 5.8

5.5 Scoping

The major problem described in this section is one of scoping. The sentence may contain collective scoping or distributive scoping. Given a sentence with ambiguous scoping, the system will display two different pictorial representations–one for a collective scoping and one for a distributive scoping.

(37) Two policeman shot seven criminals.

The collective reading is that the two policemen, together, did the shooting. The distributive reading is that each policeman shot seven criminals, for a total of fourteen criminals that were shot.

The next two subsections look at the LF for each of the sentences, the pictorial generation process for scoping, and the resulting picture. The final subsection takes a look at other quantifiers.

The logical form representation to show scoping is described in [Alshawi 90].

> Quantifiers (in the distributive case) are taken to be predicates on two cardinalities, the number of entities, *R*, satisfying the restriction, and the number of entities, *I*, satisfying the conjunction of the restriction and the body. Abbreviations for these predicates are used for common sense cases, so, for example, *for all* abbreviates λRλI.R = I, which in our notation is R^I^[eq,R,I].

Distributive Scoping

The logical form for the distributive reading of sentence 37 is:

```
%
% Reading #1
% Distributive reading
%
% Each of the two policeman shot 7 criminals
%  for a total of 14 criminals
%

quant(R^I^[eq,I,2],
      A,
      [policeman,A],
      quant(K^L^[eq,L,7],
            B,
            [criminal,B],
            [past, quant(exists,E,[event,E],
                         [shoot,E,A,B])])))
```

The pictorial representation for the distributive reading is shown in figure 5.9

Collective Scoping

The logical form for the collective reading of sentence 37 is:

```
%
% Reading # 2
% Collective Reading
%
% Two policemen (together) shot seven criminals.
%

quant(set(R^I^[eq,I,2]),
      A,
      [policeman,A],
      quant(K^L^[eq,L,7],
            B,
            [criminal,B],
            [past, quant(exists,E,[event,E],
                         [shoot,E,A,B])])))
```

The pictorial representation for the collective reading is shown in figure 5.10.

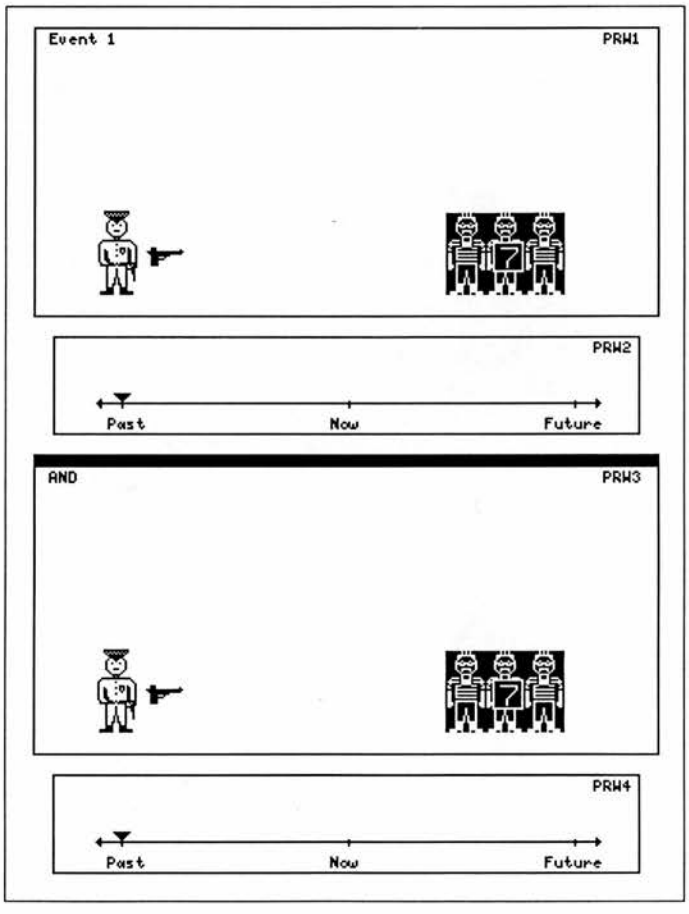


Figure 5.9: Distributive Scoping PRW

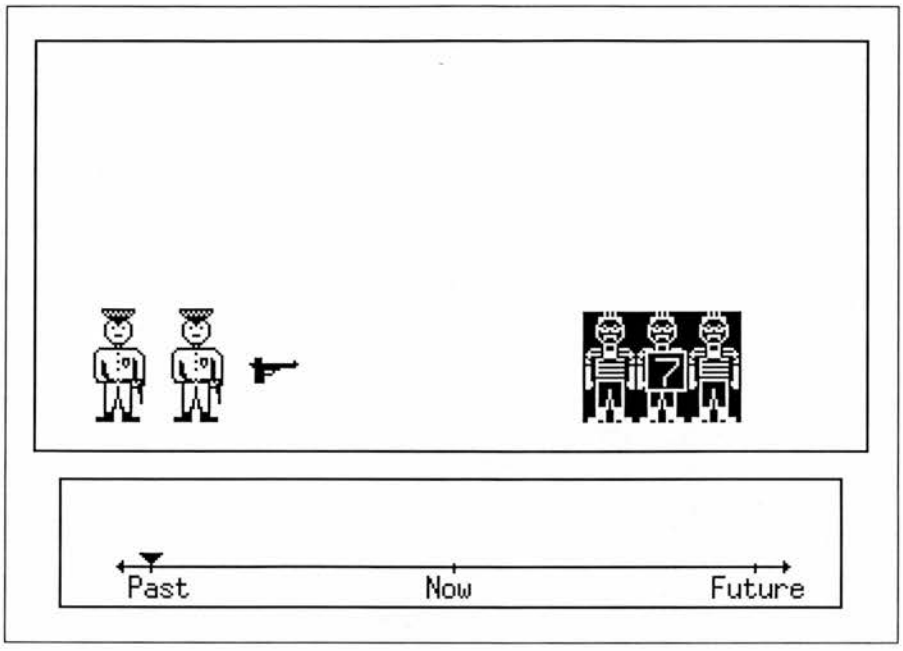


Figure 5.10: Collective Scoping PRW

5.6 Verb Features

The verb structure contains a lot of information which we would like to pictorially represent. The voice, the tense, the class of verb being used could give some information about how to pictorially represent it.

Tense information is described in Chapter 7, "Temporal Expressions." The class of verb was not implemented in this project. However, an extension to this system in the future could make use of verb classes, whether it shows movement, whether it involves objects touching or not, and whether it changes the object visually. Schank's conceptual dependency could give some clues as how to implement the verb classes.

One feature of verbs was making use of Subject-Verb-Object structure. The system uses as a default in the absence of other information, to place the subject to the left of the picture, the verb in the center (usually close to the subject) and the object to the right of the picture. Also, the the focus of the verb must be identified. It is identified by using reverse video. In figure 5.11 the cat is in reverse video because the 'cat' is the focus of seeing the cat on the table, rather than the table under the cat.

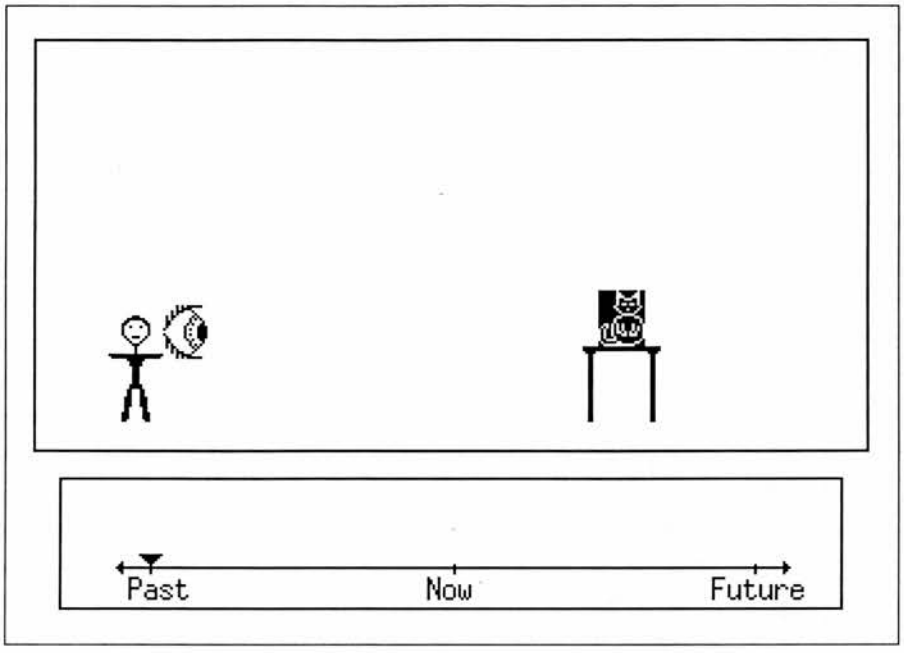


Figure 5.11: PR using SVO Structure

The default position for the subject, verb, and object was chosen for several reasons. First, the positions of left(subject), center(verb), object(right) were chosen because it matched the SVO order, which is common in English declarative text. It also did not appear to cause any problems when used with the test subjects who used the system. None of the test subjects suggested that the default position should be reversed or changed. In a "static" pictorial representation, it may be difficult to identify the actor of the sentence. Therefore, using some default location was thought to be better than random positioning of the actor, which could be confusing to the viewer, and may cause false implicatures. Another reason for using the SVO pictorial default is that it adds a few more constraints which reduce the large number of legal solutions, which helps the Constraint Satisfaction Problem Solver to find a single solution. Finally, the SVO structure is only used as a default case, in the absence of other information that forces the actor or object into particular positions. A related discussion is presented in section 6.4, "Selecting Viewpoint."

5.7 Negation

Negation is difficult to represent pictorially. In fact, Jerry Fodor stated in a televised interview, that negation could not be handled pictorially. He stated that the only option was to place a "red X" over the picture, and that that there would be a problem if one actually wanted to represent a picture containing a red X.

If one assumes that the pictorial representation is an iconic representation, rather than a photograph, then I think placing a red X over a representation would be acceptable, given that the red X was appropriately defined as the negation operator. Also, if a restriction was made to some domain, that presumably did not have red X's in it naturally, then the argument against the red X has less validity.

I propose to use a reverse video of a pictorial representation window (PRW). The reverse video is analogous to the red X, however, since the pictorial representation is making use of PRWs, then only the appropriate pictorial representation window is negated, representing the correct scoping of the negation.

Sentence 38 contains a negation in an embedded sentence.

(38) Alan said that Ian didn't see the woman on Tuesday.

An example of negation is shown in figure 5.12. Notice that only the embedded sentence PRW is negated (in reverse video) where the top level statement PRW is not negated.

There are several interpretations of this ambiguous sentence 38. The pictorial representation above represents the reading that "Alan said, at sometime, that Ian on Tuesday, did not see the woman."

The negation of an event is contrasted to negation of number, that was discussed earlier in section 5.3.1. Sentences 39 and 40 illustrate the case where an event is negated and where the number is zero.

(39) A policeman didn't see a burglar.

(40) No policeman saw a burglar.

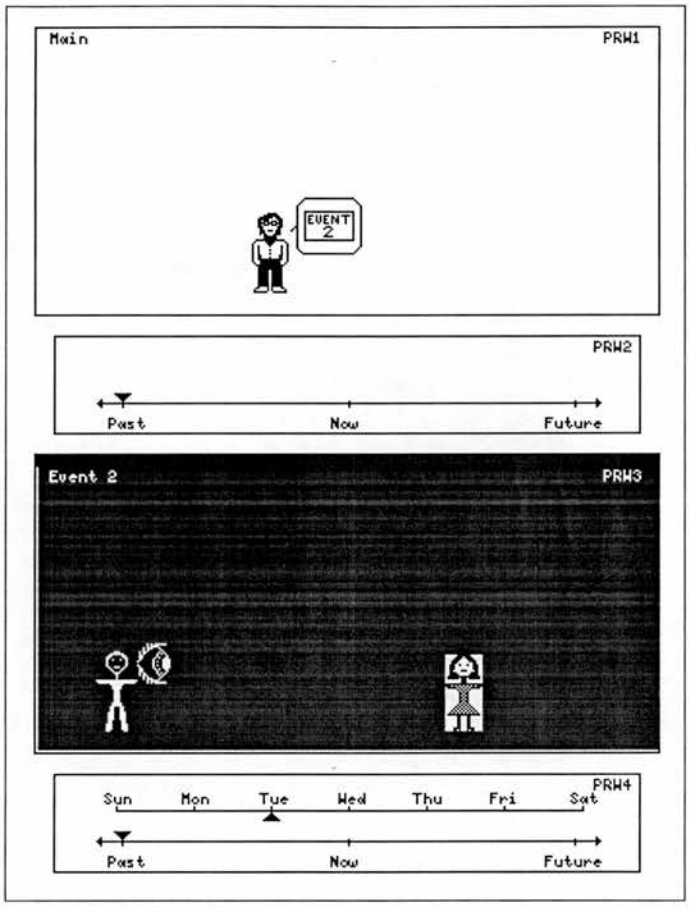


Figure 5.12: PR of Negation

The following figure 5.13 shows the two respective pictorial representations for sentences 39 and 40. This example is meant to illuminate the contrast between these two sentences.

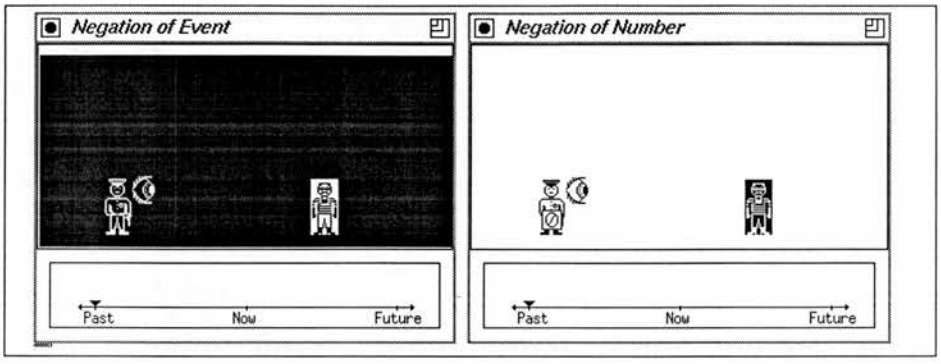


Figure 5.13: Negation of Verb vs Negation of Number

Negation of temporal expressions is discussed in Chapter 7, "Temporal Expressions."

Chapter 6

Spatial Expressions

This chapter discusses the pictorial representation of spatial expressions in natural language. When using the term **spatial expression**, I mean a natural language expression that describes an object in relation to the surroundings or other objects in terms of space. For example, an object A could be *above* object B in space. Another example is object C is *near* object D.

The previous chapter showed how several linguistic expressions were represented pictorially. However, prepositional phrases (PPs) have not been previously discussed. Many PPs are spatial expressions and generating a picture to represent these expressions seems to be an intuitive method of representation. As seen in Chapter 4, "The System Design," the technique of representing spatial expressions involves building a set of constraints to place the icons within a spatial pictorial representation window. The spatial expressions are translated into the set of constraints that are made of pictorial primitives.

This chapter will present the set of spatial pictorial primitives used in the Text–To–Pictures System. The set of primitives is defined in a two–dimensional coordinate space. A quick look at other primitives in three–dimensional coordinate space, what would be required to modify the system from two dimensions to three, and at how the CLARE logical form represents spatial expressions are given. Two classes of spatial prepositions (topological and projective) are presented, with their definition in terms of pictorial constraints and primitives, and their associated pictorial representation. A discussion about spatial descriptors is given. The chapter concludes with a look at Herskovits's Spatial Elementary Concepts, and how those concepts can be described more generally

in pictorial terms.

6.1 Pictorial Primitives for Spatial Expressions

There are several knowledge sources that contain constraints for the pictorial generation process. At the lowest level these constraints are defined in terms of a set of primitives. Examples of such primitives are: above, below, left-of, right-of, et cetera.

The primitives use a constraint notation used within the CHIP (Constraint Handling in Prolog) language. The '#' symbol preceding any of five equality operations represent a constraint on that variables legal domain. The symbols '#=', '#>', '#<', '#>=', '#<=' represent constraints of equal-to, greater-than, less-than, greater-than-or-equal-to, less-than-or-equal-to, respectively. If a variable A has a range of acceptable solutions between 0 and 100 (notation of [0..100]), and the variable is constrained by A #< 50, then the legal domain is reduced to [0..49].

Most sentences can be described with primitives that operate within a two–dimensional coordinate space. However, there are a few expressions that require three–dimensional space, in order to be properly pictorially represented. The next two sections describe the pictorial primitives in two and three-dimensional coordinate space respectively.

6.1.1 Two–Dimensional Coordinate Space

The **two–dimensional coordinate space** is a grid defined by a tuple of variables (x, y), where the X–axis is along the horizontal, and the Y-axis in the vertical direction. This space assumes a fixed viewpoint at ground level looking towards the objects, with all objects in view. For example, the intuitive meaning of the "above" operation would be translated to a primitive *above* that is defined as bottom of object one having a greater Y–axis value than top of object two.

A partial set of the Spatial Expression Pictorial Primitives was given in section 4.2.2, to show an example of how the system works. This section presents the full set of Spatial Expression Pictorial Primitives. The constraints (or inter–relationships among objects) are made up of combinations of these primitives. The following primitives all involve a relation between two objects, while one primitive involves three objects. Considering

objects A, B, and C, the set of spatial expression pictorial primitives is:

	primitive	*description*
1	above(A,B)	A is above B
2	below(A,B)	A is below B
3	left-of(A,B)	A is to the left of B
4	right-of(A,B)	A is to the right of B
5	touching(A,B)	A touches B
6	near(A,B)	A is next to B
7	far-from(A,B)	A is far from B
8	between(A,B,C)	A is between B and C
9	in-top-third-region(A,B)	center of A is in the top third region of B
10	in-center-vert-region(A,B)	center of A is in the center third region of B
11	in-bottom-third-region(A,B)	center of A is in the bottom third region of B
12	in-left-third-region(A,B)	center of A is in the left third region of B
13	in-center-horiz-region(A,B)	center of A is in the center third region of B
14	in-right-third-region(A,B)	center of A is in the right third region of B
15	overlap(A,B)	A overlaps B

- Primitives 1 through 4 are directional constraints that define a region where one object is allowed to be placed in relative position to the other object.
- Primitives 5 through 7 are subjective constraints that are dependent upon distance. All three of these primitives are defined using the same technique, but have varying distance values. The distance value for 'touching' is zero. The values for distances associated with 'near' and 'far-from' can be modified by the user. Experimentation was used to define these distances.
- Primitive 8 is a constraint involving both distance and region relations. Object A must be placed in a region defined by objects B and C.
- Primitives 9 through 14 describe regions within an object. The center of object A is placed within a region that is defined by dividing object B into thirds.
- Primitive 15 involves looking at the area of two objects. This constraint is disjunctive. The area of object A is constrained to overlap part of the area of object B. Touching is not considered overlapping.
- The "not" operator can apply to each of these fifteen constraints. *Not(next-to) and far-from* are different primitives.

Above, Below, Left-Of, Right-Of Primitives

The primitives for *above*, *below*, *left-of*, and *right-of* constrain one object to be in a region relative to the position of another object. Object B is the point of reference in figure 6.1. The shaded region is the area that is *left-of* object B.

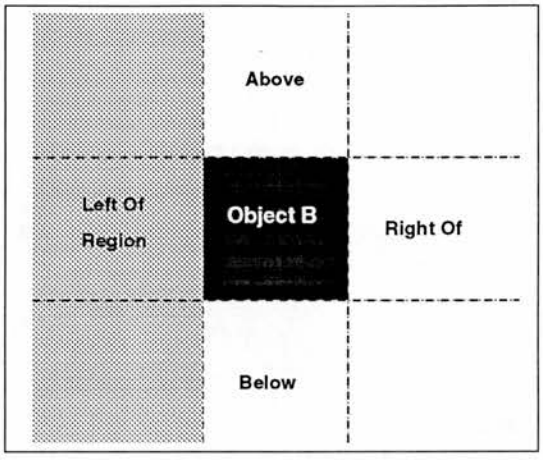


Figure 6.1: Above, Below, Left, Right Regions

To constrain an object A to the shaded region, the *left-of* primitive is applied. The *left-of* primitive is defined as:

```
%
% Object A is ''left of'' Object B
%
in-left-of(
   [_ObjID_A,_Tag_A,_Left_A, Right_A,_Bottom_A,_Top_A,_Front_A,_Back_A],
   [_ObjID_B,_Tag_B, Left_B,_Right_B,_Bottom_B,_Top_B,_Front_B,_Back_B])
% the 'right side' of object A touches or is to left of the
% 'left side' of object B.
   Right_A #<= Left_B.
```

The other three primitives (*above*, *below*, and *right-of*) are defined in much the same way.

Touching, Near and Far-From Primitives

An operation that can constrain object B to be in a space surrounding object A is assumed. In other words, this space can contain object B in any position surrounding

object A, so that it is touching, but not overlapping. In any position for object B, the distance from Object A to Object B is zero. Figure 6.2 shows this region around object A. This definition makes use of not overlapping, which will be described later in this section.

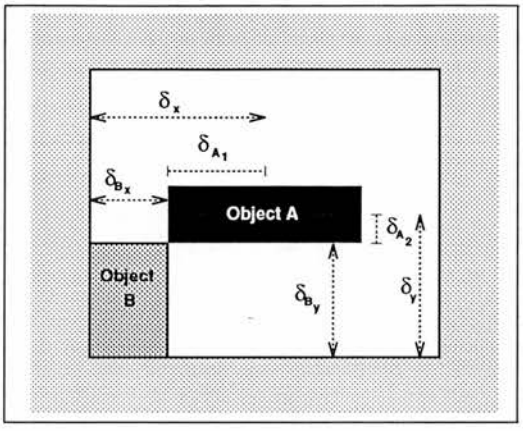


Figure 6.2: Touching Region

This region is determined by computing the distances δ_x and δ_y, which are the distances from the epicenter of object A to the edge of the bounding region along the respective x or y axis. Object B is then constrained to be within that distance.

The value for δ_x and δ_y is calculated by adding the distance from the epicenter of object A to its appropriate edge (half-length or half-height), plus the length or width of object B, plus a user-defined distance ϕ.

Given:

δ_{A_1} is the half-length of Object A,

δ_{A_2} is the half-height of Object A,

δ_{B_x} is the length of Object B,

δ_{B_y} is the height of Object B.

Then:

$$\delta_x = \delta_{A_1} + \delta_{B_x} + \phi,$$

$$\delta_y = \delta_{A_2} + \delta_{B_y} + \phi.$$

Each of the four sides of the region must be constrained[1]. For example, the left side of the region would be defined by stating that left side of Object B must touch or be to the right of that edge of bounding region. Given that the epicenter of Object A is (ξ_x, ξ_y), the sides of the bounding region are constrained by:

$$\begin{aligned}
&\text{Left_B } \#>= \quad \xi_x - \delta_x \ \wedge \\
&\text{Right_B } \#>= \quad \xi_x + \delta_x \ \wedge \\
&\text{Top_B } \#<= \quad \xi_y + \delta_y \ \wedge \\
&\text{Bottom_B } \#>= \quad \xi_y - \delta_y.
\end{aligned}$$

In the example of *touching*, $\phi = 0$. To define, the *near* primitive, ϕ is set to some small value. This will then define the region to be as shown in figure 6.3.

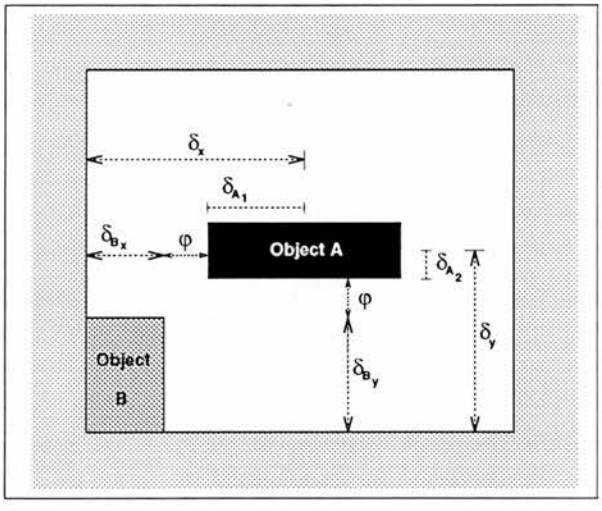


Figure 6.3: Near and Far-From Regions

The *far-from* primitive is different from the *near* primitive is that ϕ is large value, and that Object B must be outside of this bounded region. To constrain B to be outside of the region, a disjunctive definition is required. Disjunctive constraints are discussed at the end of this section in the discussion about the *not* operator. Although the *far-from* primitive is disjunctive, the bounding region is determined using the same technique as the *near* primitive.

[1] This definition is for objects that are strictly rectangular. However, for computational reasons, the system also uses this assumption for objects that are not rectanglar.

Between Primitive

The *between* primitive constrains an object A to be "between" objects B and C. This is done by constructing a rectangular boundary from the two closest corners of objects B and C. This region is shown in figure 6.4. However, it is possible that object A is bigger than the between region that is defined. This primitive only constrains the epicenter of object A to be within the "between region."

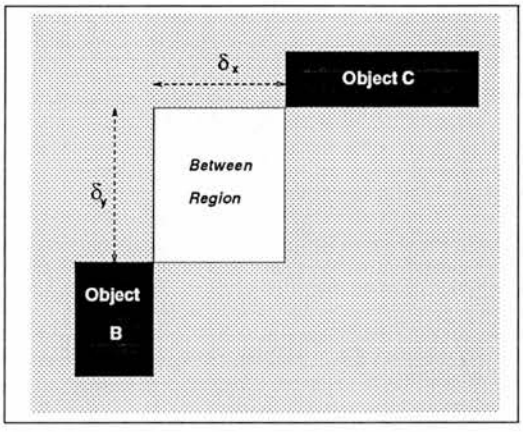


Figure 6.4: Between Region

Given that the epicenter of Object A is (ξ_x, ξ_y), and if objects B and C are placed as they are in figure 6.4, then the "between region" can be constrained by:

$$\xi_x \# >= Right\_B \ \wedge$$
$$\xi_x \# <= Left\_C \ \wedge$$
$$\xi_y \# >= Top\_B \ \wedge$$
$$\xi_y \# >= Bottom\_C.$$

In-a-Third-Region Primitives

We have seen how objects are constrained by primitives that force an object to some position *outside* of the reference object. Now the primitives that operate *within* an object are presented. An object is divided into three equal divisions horizontally and three equal divisions vertically. These dividing lines can extend beyond the actual object to define a region. The six regions defined by divisions of the object are shown in figure

6.5. Note, that these six primitives define more than the nine squares. If an object is in the *left-third region*, it could be in any of the three squares on the left, could be above the object or below it. A primitive only constrains an object to be located in that *region*. With combinations of primitives a specific location can be defined.

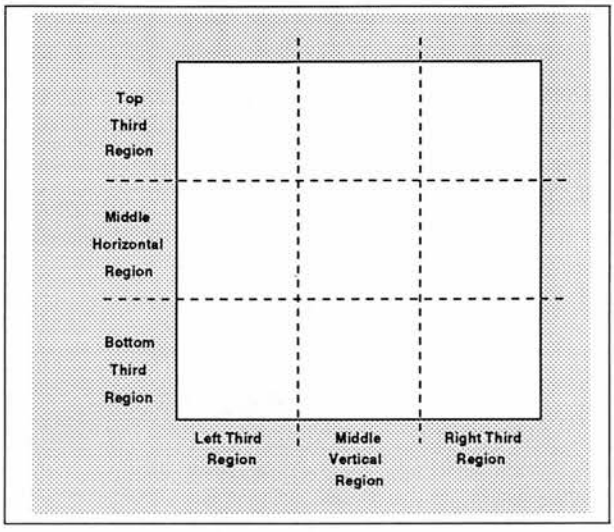


Figure 6.5: In-Third-Of Regions

The *in-a-third-region* primitives constrain the epicenter of object A to be bounded by the appropriate in-a-third-region of object B. In figure 6.6, there are two objects that are in the the top–third region of Object B. Object A_2 is located entirely within the region defined. Object A_1 is large in height, and can not be completely bounded within the region. However, the epicenter is constrained to be within the region, which is all that is required by this primitive.

κ_{B_y} is defined as one-third of the height of object B. This is determined by:

$$\kappa_{B_y} = \frac{Top\_B - Bottom\_B}{3}$$

Given that the epicenter of Object A is ξ_x, ξ_y, the definition of the *in-top-third-region* primitive is:

$$\xi_y \ \# <= Top\_B \ \wedge \ \ \xi_y \ \# >= Top\_B - \kappa_{B_y}.$$

The other five primitives *in-bottom-third-region*, *in-middle-horiz-region*, *in-left-third-*

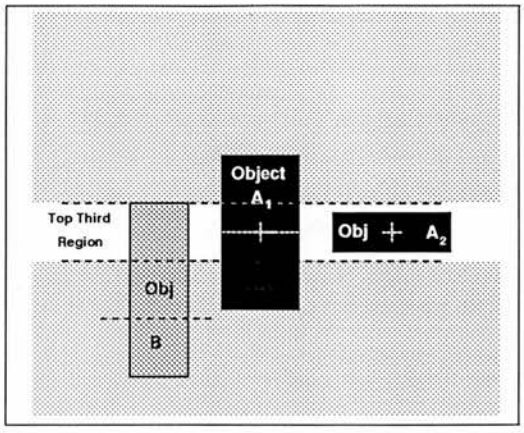


Figure 6.6: In-Top-Third Region

region, *in-middle-vert-region*, and *in-right-third-region* are defined in a similar method.

Overlapping Primitive

The *overlapping* primitive requires that some part of object A "overlaps" object B. This could be easily forced by requiring the epicenter of A to be constrained by the boundary of object B itself. However, this seems to be too restrictive of an approach, because there are other possibilities of object A barely overlapping object B. This is important because this primitive is most useful in the negated sense, *not(overlap(A,B))*. This would constrain that object A cannot overlap B in any way. This requires a disjunctive constraint.

Figure 6.7 shows that objects can be of varying sizes. Checking that two points of Object A1, are on different sides of an edge of Object B1, would tell us that the object does overlap. However, Object A2, does not cross any edge boundary of object B2, because A2 is completely contained within object B2. Object A3 overlaps B3 in the vertical, however if the test was reversed to look for B3 crossing an edge of A3 in the vertical, the test would fail. The same is true for A4 and B4 in the horizontal. The only way of adequately constraining that one object does not overlap another object is to disjunctively constrain it to be either left-of, right-of, above, or below. The disjunctive constraint for *not(overlap(A,B))* is:

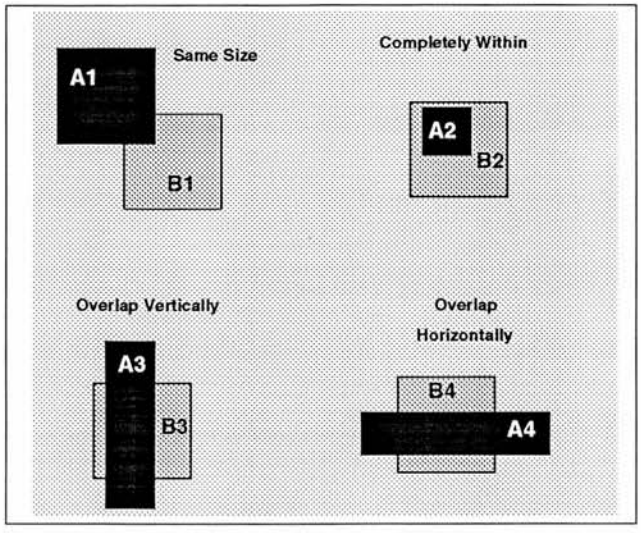


Figure 6.7: Overlapping Regions

$$Left\_A \ \# >= Right\_B \ \vee$$
$$Right\_A \ \# <= Left\_B \ \vee$$
$$Bottom\_A \ \# >= Top\_B \ \vee$$
$$Top\_A \ \# >= Bottom\_B.$$

Negation of Primitives

Each of the primitives must allow the not operation. One concern, when applying the not operation to a primitive, is that a constraint may become disjunctive. The functions *left-of*, *right-of*, et cetera can be defined conjunctively. However, any of the primitives that constrain an object to be inside a bounding region will require a disjunctive operation to constrain the object to be outside the region. This is not a problem, but the conjunctive constraints are preferable in that it is very easy to see each constraint reducing the legal solution for a variable. With a disjunctive constraint, multiple solutions could be offered, and it may not be obvious that one of disjunctive solution paths is illegal until several other constraints have applied. Therefore in the implemented system, every effort was made to use conjunctive constraints, and to try to apply the disjunctive constraints only after the conjunctive constraints are exhausted. This worked well with the *not(overlap)* constraint being applied after the other constraints. The only other disjunctive constraint was the *far-from* constraint. In the system, a large value for ϕ was used, so the bounding

region was quite large. Quite often three of the four sides were not within legal boundaries for a variable, and thus only one acceptable solution path was offered. This in a sense, made the constraint act in a conjunctive way.

The set of primitives that will be able to pictorially represent two objects in various combinations of locations in a two–dimensional space were shown. The next section shows how to extend this to a three–dimensional coordinate space.

6.1.2 Three–Dimensional Coordinate Space

The **three–dimensional coordinate space** is a cube that is defined by a triple of variables (x, y, z), where the X–axis is along the horizontal, and the Z-axis in the vertical. The Y-axis represents depth into the two–dimensional coordinate space. The three-dimensional coordinate space described does not assume a predetermined viewpoint.

Three-dimensional representation would only be needed when two-dimensional representation is not sufficient. This is true for the spatial expressions *in front of* and *behind.* With these expressions, a three-dimensional picture must be used to represent the scene.

	primitive	*description*
16	in-front-of(A,B)	A is in front of B
17	behind(A,B)	A is behind B

three-dimensional coordinate space was not implemented in the system described in this thesis. This was because of a lack of a three-dimensional high level graphics system to interface with Prolog. The conversion of the two-dimensional primitives to three dimensions would basically involve inclusion of a third value to represent the third dimension. Since most of the two-dimensional primitives do not require any operations in the third dimension, there would be no modification of those primitives.

The *in-front-of* and *behind* operations are similar to the *left-of* and *right-of* operations, except that they are three-dimensional. If implemented, an example of a three-dimensional primitive would be:

```
%
% Object A is ''in front of'' Object B
```

```
%
in-front-of(
   [_ObjID_A,_Tag_A,_Left_A,_Right_A,_Bottom_A,_Top_A,_Front_A, Back_A],
   [_ObjID_B,_Tag_B,_Left_B,_Right_B,_Bottom_B,_Top_B, Front_B,_Back_B])
% the 'back' of object A is equal to or in front of the 'front' of
% object B
   Back_A #<= Front_B.
```

Now that we have the complete set of Spatial Expression Pictorial Primitives, let's look at how spatial expressions are described in the CLARE logical form representation, and how the Text-To-Pictures System bridges the gap.

6.2 How the LF Represents Spatial Expressions

The CLARE logical form represents spatial expressions similarly to how it represents verbs[2]. Spatial prepositions are defined as having one or more *senses*. The functor (the preposition) is listed with its arguments (the objects).

(41) There is a cat on the table.

Given sentence 41, the generated logical form shows the preposition "on" in the *on Locational* sense. It relates two objects B and C, and is read as "object B, in the spatial sense, is located *on* object C."

```
[dcl,
 quant(exists,
       A,
       [entity,A],
       quant(exists,
             B,
             [and,
              [cat_Animal,B],
              quant(exists,C,[table_Furniture,C],[on_Locational,B,C])],
             quant(exists,D,[event,D],[be,D,A,E^[eq,E,B]])))]
```

An example of a small portion of the CLARE file that defines the preposition senses is listed below. Note that several of the prepositions have temporal senses, which have as the last argument "temp." The spatial senses are marked with "loc" in the last argument.

[2] Section 2.3.3 describes how to read the CLARE logical form structure.

```
prep_sense(after,after_Temporal,temp).
prep_sense(against,against_Locational,loc).
prep_sense(at,at_Locational,loc).
prep_sense(at,at_Temporal,temp).
prep_sense(before,before_Temporal,temp).
prep_sense(before,before_Locational,loc).
prep_sense(beside,beside_Locational,loc).
```

CLARE has a rather exhaustive list of preposition senses. However, it is possible that a sense intended in a sentence, is not contained in the CLARE list of prepositions senses. A logical form is only generated for the preposition senses that are listed. However, it is relatively straight forward for a user to add preposition senses if he or she wishes.

The preposition senses are not defined in any further terms within the logical form. The preposition sense is further defined by the pictorial grammar in terms of a set of constraints. For example, the *beside.Locational* preposition sense is described as:

```
spatial_defs(['beside_Locational',A,B],OldCS,NewCS) :-
   NewCS = [
            not(above(A,B)),
            not(below(A,B)),
            near(A,B)
            | OldCS], !.
```

This call adds three constraints to the total constraint list. It says that object A is not above, nor below, object B, and that the two objects are close to each other.

6.3 Spatial Prepositions

Given the set of primitives and how the logical form represents spatial prepositions, several pictorial representations for a sample of spatial expressions are presented. Two types of prepositions [Herskovits 86], topological and projective, will be looked at. **Topological prepositions** are locative expressions that describe the location of one object relative to another object in terms that are not affected by the view point. Topological prepositions include *in*, *at*, and *on*. **Projective Prepositions** are prepositions that express some directional information, about how an object is located with respect to another object, that is dependent upon the viewpoint. These include *behind*, *left of*, *beside*, et

cetera. These prepositions are dependent upon the viewpoint. The notions of viewpoint and naive physics, such as gravity, are discussed fully in the two sections that follow this section on spatial prepositions.

6.3.1 Topological Prepositions

This section shows two examples of topological prepositions: *on*, and *in*.

Example of "On"

Chapter 4 showed a detailed example of how a pictorial representation is generated for sentence 42:

(42) The man saw the cat on the table.

This sentence generates two logical form descriptions. The one that represents the cat being located on the table (rather the man standing on the table, and he sees the cat somewhere else) is shown in figure 6.8.

The definition of the preposition *on*, that is used in this thesis, was defined previously in Chapter 4 as:

```
on_location(A,B) :- above(A,B),
                    touching(A,B).
```

Several different definitions were tested for the preposition 'on', however, the definition shown above seems to be the most succinct. An earlier definition stated that the object was located *above*, and *not(left)* and *not(right)* to force it to be directly above. But that could be too restrictive, and uses an extra primitive. The definition chosen is motivated by that object A must *touch* the object it is *on*, and also be above it. The pictorial representation using this definition is shown in figure 6.9.

```
%  1st Logical Form of:
%    "The man saw the cat on the table."
%

[dcl,
 quant(exists,
       A,
       [man_MalePerson,A],
       quant(exists,
             B,
             [cat_Animal,B],
             quant(exists,
                   C,
                   [table_Furniture,C],
                   [on_SpatiallyOnTopOf,B,C])],
       quant(exists,D,[event,D],[past,[see_LookAt,D,A,B]]))]
```

Figure 6.8: Logical Form for {{The man} saw {{the cat} on {the table}}}

Example of "In"

A team at University of Toulouse [Aurnague & Borillo 90a, Aurnague & Borillo 90b, Borillo & Borillo 90, Vieu & Borillo 90, Vieu 91] has done extensive work on the preposition *in* or *dans*[3]. Their work was quite in depth, and had several rules of describing a set of spatial semantics. [Vandeloise 86] has also described the preposition *dans* in depth. However, the Toulouse team describe Vandeloise's description as "perhaps too synthetic...his point of view is all-functional and make use of *family resemblances.*" They describe Herskovits's work in the area as "detailed description but not synthesized logically."

The Toulouse team identify three categories of *dans*:

- Total *dans* ('ds')
- Partial *dans* ('dsp')
- Part-of *dans* ('ds/pt-of')

[3] *Dans* is roughly the French equivalent of the English *in*.

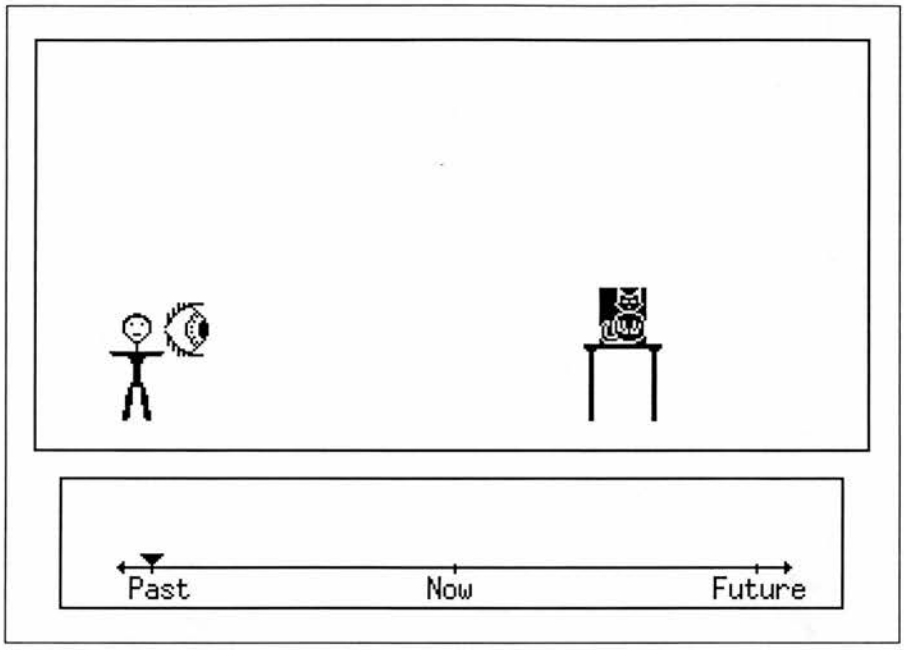


Figure 6.9: PR of *on*

Three example sentences of their divisions are:

(43) *Le vase est dans la valise.*
The vase is in the suitcase.

(44) *Le bouquet est dans le vase.*
The bouquet is in the vase.

(45) *La fleur est dans le bouquet.*
The flower is in the bouquet.

Borillo et al draw a distinction between the first two cases, as in sentence 43 the first object (the vase) is entirely contained within the object two (the suitcase). In sentence 44 the first object (the bouquet) is only partially contained within the second object (the vase).

[Vieu & Borillo 90] gives a spatial semantic to describe a *partial* function. This function describes when one object is partially contained within another. Sref is a spatial description in reference to some object in the parenthesis.

$$A'dsp'B \equiv \exists i, Sref(part(A,i)) \subset Sref(whole(B))$$
$$\wedge\ Sref(A) \neq Sref(B) \wedge not(B'ds'A)$$

In a pictorial representation, it appears that making the distinction of whether an object is entirely within an object or not, is not required. Using naive physics, and a simple definition of *in*, I will pictorially represent an object partially in another object.

Knowing that our picture will be in a two-dimensional space, some ingenuity will be required to display the *in* preposition. The goal is to see object A and to see it surrounded by object B. Given a fixed viewpoint in two-dimensional space, this can be accomplished by constraining object A to overlap object B. Therefore, the definition of the preposition *in* is:

```
in_location(A,B) :- overlap(A,B).
```

However, object A needs to be contained within object B. At first thought this can be accomplished by constraining object A to not be above object B in any way, to not be below, left, nor right. As previously seen, it is possible that an object may be *in*, but may also protrude, extending outside of the region of object B. Sentence 46 shows an example of where the object A (the plant) may extend outside object B.

(46) The man saw the bouquet in the vase.

From a pictorial view, objects that are containers, may have an opening where objects can protrude outside of their container. In this case, an object which is shown to be a container (the vase), has an opening at the top. Therefore, one wants to constrain the sides of the container only. So, object B (the vase), has as its definition that other objects cannot overlap *and* be also be below, to the left, or to the right.

If object B is a container, then the constraint, *in_container* is added by the Naive-Physics Module. The constraints for *in_container* are defined as:

if B has a top side, then add constraint not(above(A,B)),
if B has a bottom side, then add constraint not(below(A,B)),
if B has a left side, then add constraint not(left(A,B)),
if B has a top side, then add constraint not(right(A,B)).

Given, that a vase has a bottom, left, and right side, the set of constraints for in_location is expanded to:

```
in_location(A,B) :- overlap(A,B),
                    not(below(A,B)),
                    not(left(A,B)),
                    not(right(A,B)).
```

Given this situation we, want to draw object A first, and then have object B *overlap* it. As a general rule, where one object is hidden from view, that object is drawn using a dashed line to show that it exists, but is hidden from view[4].

The pictorial representation for sentence 46 is shown in figure 6.10.

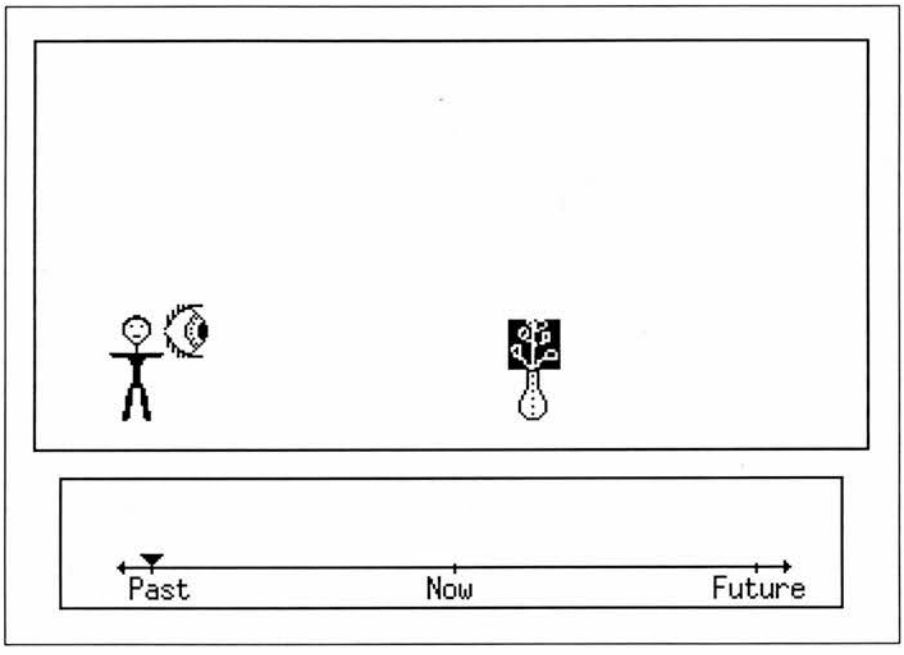


Figure 6.10: PR of *in*

6.3.2 Projective Prepositions

Two projective prepositions will be pictorially represented in this section: *beside*, and *left of*. Viewpoint is especially important when handling projective prepositions. Both of these prepositions are shown using the system's two-dimensional space. The problem of selecting viewpoint is discussed fully in the next section.

[4] This is a standard architecture drawing technique.

Example of "Beside"

This preposition relationship constrains one object to be located next to another object. For a pictorial representation, preferably we would see both objects. This rules out viewpoints where the one object is in front of, or behind the other. The viewpoint will be assumed to be looking from the speaker. The beside preposition should constrain the pictorial representation to draw object A near object B, but not actually touching each other. If they touched, that might imply another relationship that was not intended. Also, when one thinks of this relationship in pictorial terms, one does not think of one of the objects above or below the other object.

The definition of the preposition *beside* is:

```
beside_location(A,B) :- near(A,B),
                        not(below(A,B)),
                        not(above(A,B)).
```

(47) The man saw the cat beside the table.

The pictorial representation for sentence 47 is shown in figure 6.11.

Notice that the cat is emphasized in reverse video to indicate that the cat is the object of the seeing event, and not that table is the object with the cat next to it.

Example of "Left-Of"

The *left of* preposition has the basically the same problem of viewpoint that *beside* has. Again, one wants to see both objects involved in this relationship, and the picture should be oriented as from the speaker.

One doesn't want to overconstrain the location of object A. It should be located somewhere to the left of object B.

Our definition of the preposition *left of* is:

```
leftof_location(A,B) :- left(A,B),
                        not(touching(A,B)).
```

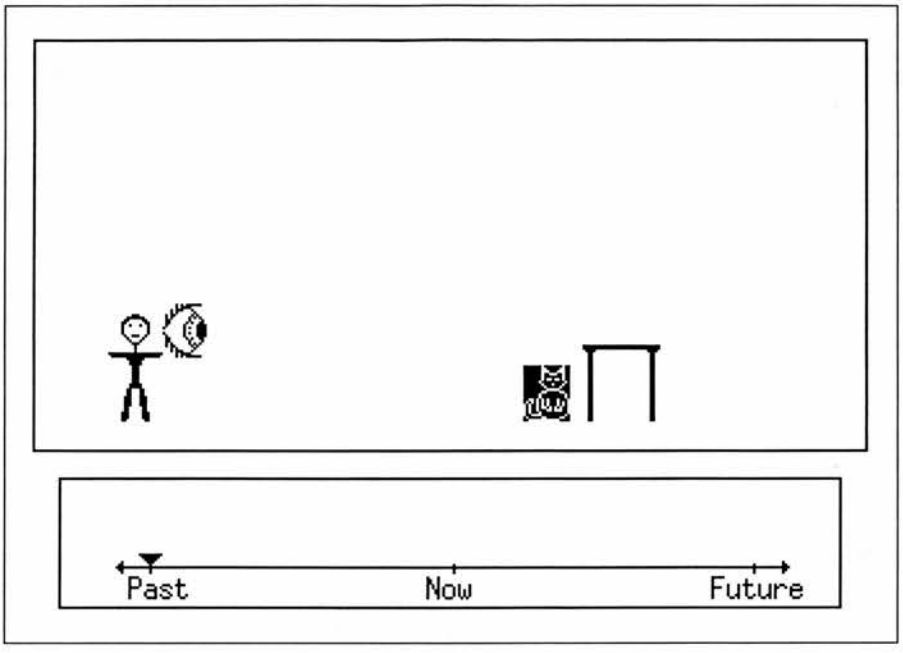


Figure 6.11: PR of *beside*

Given sentence 48,

(48) The cat is to the left of the table.

the pictorial representation is shown in figure 6.12.

Example of using "In-Top-Third-Of"

The *in-top-third-of* primitives are used when describing a specific location within one object, or to gain precision in locating two objects relative to each other. For example, the verb *see.LookAt* specifies that the eye icon to represent the seeing event is located in the top third region of the person doing the seeing.

The see_LookAt verb is defined as:

left(Actor,Event),
touching(Actor,Event),
in_top_third(Event,Actor),

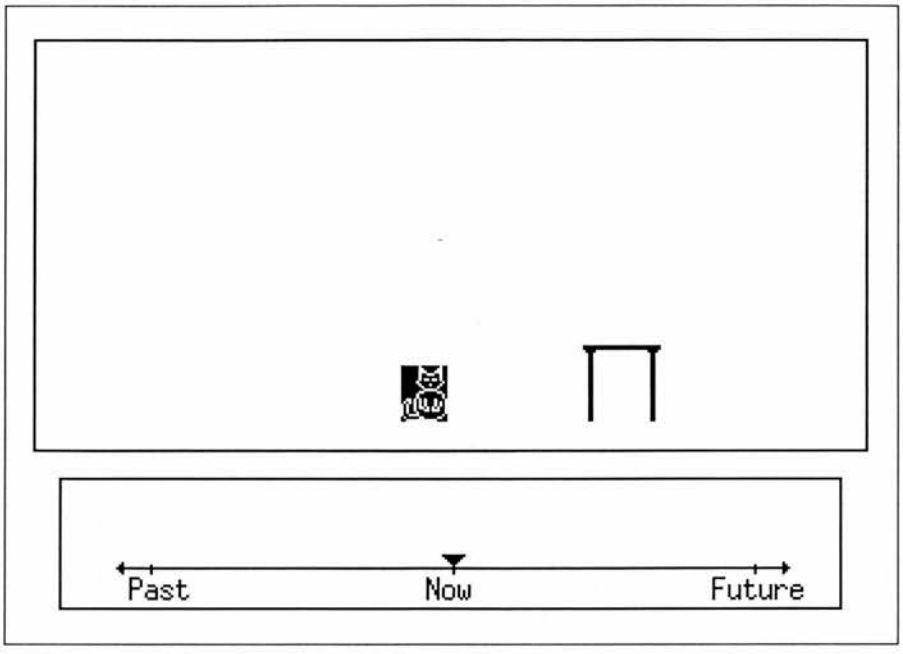


Figure 6.12: PR of *left of*

left(Event,Patient),
farfrom(Event,Patient)

The third constraint is that the epicenter of the event icon (the seeing icon) is located in the top third region of the actor icon. The following figure shows the seeing icon in the top third region of the man on the left. The seeing icon is centered on the man in the middle, and is in the bottom third region for the man on the right.

6.4 Selecting Viewpoint

Selecting viewpoint is important for determining reference. In the system implemented in the thesis, the picture is drawn from a viewpoint of a third person, and the viewpoint does not move, which makes the problem of selecting the view point tractable. Although not implemented in the working system, the problem of selecting the viewpoint, and of moving or multiple viewpoints should be mentioned.

In the last two examples, the frame of reference was from a third person outside looking towards the picture. However, this could result in problems. A problem described by

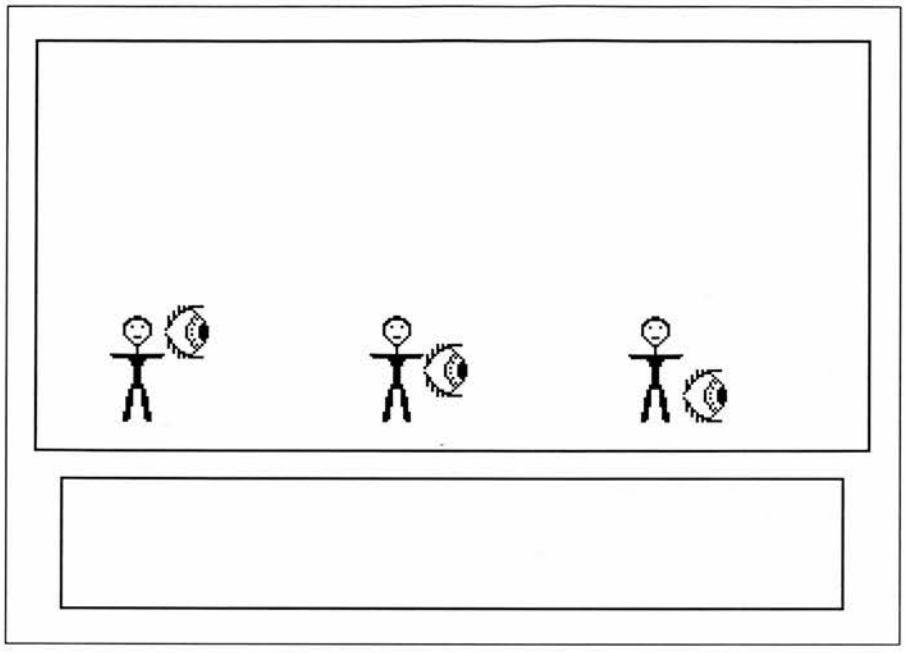


Figure 6.13: Use of *In-Top-Third* Primitive

Retz–Schmidt [Retz-Schmidt 88] is that the reference of the viewpoint changes whether one is inside or outside of the object. See figure 6.14.

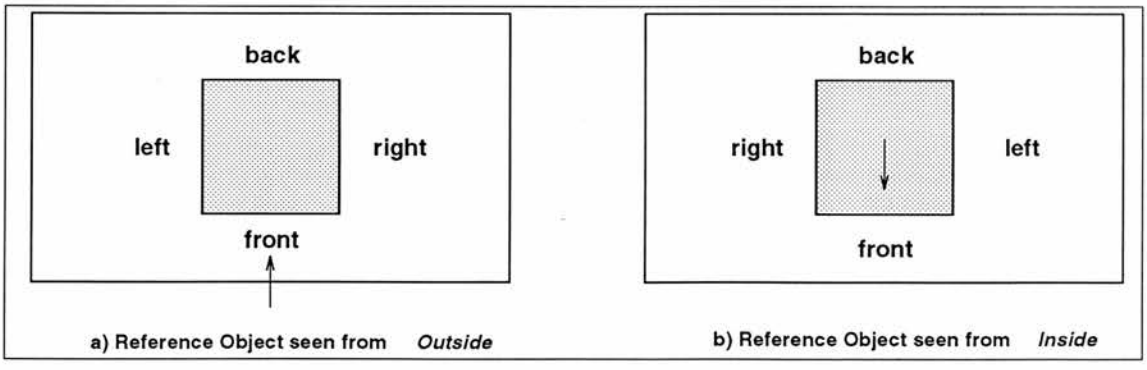


Figure 6.14: Two Reference Systems of Differing Viewpoint

Figure 6.14 is an object with its sides labeled as seen from the outside. The *front* of that object is the closest to the viewpoint, and probably in view. The left and right sides are oriented from the viewpoint. The back is furthest from the viewpoint. Where figure 6.14b is an object looking at it from the inside. The front inside wall would be directly in front of the viewer. This also corresponds to back of the object. However, notice that the

left and right positions are reversed between the two figures. These deictic expressions are dependent on the viewpoint.

The viewpoint in this system is normally from a third person. However, a special case occurs when using 'I', 'me' or speaker.

(49) Ken saw the man on the hill.

(50) I saw the man on the hill.

(51) I drove the car.

In sentence 49, the pictorial representation is from the third person point of view, looking at both, Ken, and the man on the hill. In sentence 50, one could set the viewpoint to reflect what the speaker sees. In that case, one would only see the man on the hill. When asking subjects to sketch a sentence, several people came up with a drawing from this viewpoint for sentence 50. However, this representation loses some information, in that the speaker is not shown. Sentence 51 shows a different event, and definitely both the speaker and the car should be drawn. Therefore, it was chosen to always draw from a third person perspective, showing the speaker as a pictorial object.

A particular problem may arise, when locative descriptions reference themselves rather than other objects. This prevents using the simple technique of locating objects from the third person point of view. Look at the following sentences:

(52) Maria is to the left of the hill.

(53) Ian saw Maria on *his* right.

Maria was able to placed to left of the hill, in the previous example, from a viewpoint of a third person. However, in sentence 53, the description is given from the viewpoint of Ian. The pictorial representation should still be from a third person, for reasons discussed in the last section, but Maria must be correctly placed. As a default case, in the absence of specific locative information, the actor is located to the left, and the object to the right. The objects have to be drawn in some position, and that was chosen to match the SVO structure. If one constructs a representation where Ian is facing the third person

viewer, as in sentence 53, the "actor to the left of picture" default is not used. The specific information overrides the default case. Maria is still identified as the object by the emphasis of reverse video, identifying her as the object.

Some situations really require a three-dimensional coordinate space, and make use of *multiple viewpoints*. Donia Scott [Scott 92] asked several subjects to describe Brighton airport. The descriptions had several different viewpoints. The pilots gave a "bird's eye view" looking down on the airport. Many of the ground personnel gave a moving viewpoint, as if walking on a tour through the airport. Descriptions of *left of* are ambiguous without a definite position of viewpoint. The current sentence may have a different viewpoint from the last one, because the speaker has moved through the scene to another location. Knowing the location of the speaker in the visual scene is critical to determining projective spatial expressions. The system in the thesis got around this problem by dealing with only single sentences, and thus forcing the viewpoint to *one* location.

6.5 Naive Physics

The primitives described in the last two sections are not enough to generate a pictorial representation. The Naive–Physics Module provided additional constraints to complement the preposition definitions. This may seem to be an unusual approach to describe the meanings. However, it provides a mechanism so that the preposition descriptions are succinct and intuitive. Often the constraints that are generated from the Naive-Physics Module are considered obvious. Sometimes so obvious, that when asking someone to define a preposition they leave out the *assumed* naive-physics constraints. One such constraint is gravity.

Gravity is implemented by constraining that all objects must rest upon another object (unless they can defy gravity). Further, a pictorial *ground* is introduced. The Pictorial Ground in the system provides something for the objects to rest upon if they are not resting upon another object. Objects are checked to see if they can rest upon the ground. They might not be able to rest upon the ground, because they are previously constrained, such as an object is above another object. Two objects cannot both touch the ground and one be completely above the other. If they can rest upon the ground, and are not prevented from doing so by previous constraints, then a constraint that "the pictorial

object must touch the ground" is added. If an object is above another object, and it can rest upon the object below (i.e. not prevented by previous constraints), then they are also constrained to touch.

Another concept important to spatial expressions is the **figure–ground relationship**. Several have used variations of this concept in work concerning spatial expressions [Talmy 75], [Talmy 78], [Talmy 83], [Talmy 85], [Herskovits 86], [Lang *et al* 90], [Maienborn 91].

Usually this relationship is used to show which objects can and cannot move. This is very important for determining reference.

(54) The bike is near the house.

(55) * The house is near the bike.

The pictorial representation of sentence 54 should not be different from sentence 55. However, sentence 55 is marked in English. Talmy explains this markedness of sentence 55 by applying *special semantic notions* to nominals to determine roles. The Ground (the house) is a presupposed notion, where the Figure (the bike) is an asserted notion.

Although the pictorial representations for both sentences are the same, the general concept of the figure-ground relationship becomes important when applying constraints. If object A touches object B, this involves constraining object A to be in the area surrounding object B, but does not constrain the placement of object B directly. The constraints can be thought of as bidirectional, with one exception. Usually after all the constraints have been applied, a *range* of solutions still exist. The objects must be drawn in some precise location. So the technique is to *fix* one location first, which reduces the legal range of solutions for the other objects. This technique is done until all of the objects have a precise location. The order of which objects to give a hard location first, is determined by trying to fix the object that the others refer to.

The determination of whether object A is constrained to object B, or vice versa, is first attempted by linguistic cues–an instrument is constrained to the actor. When linguistic cues offer no help, as in sentences 54 and 55, a gross simplification of the relationship is implemented which makes use of the sizes of the icons. Small pictorial objects are

constrained to the bigger pictorial objects.

The importance of the figure-ground relationship is actually of little importance in the current system. However, if objects were to move, this relationship would be critical. Therefore, the concept was introduced into the Text-To-Pictures System.

Also the concepts of Gravity may seem rather intuitive, however, if they are not implemented, the resulting set of constraints does not require that the objects be oriented towards the ground. With the concept of gravity not implemented, and the objects placed satisfying the other constraints, often the objects were placed incorrectly above and below other objects causing a false implicature– indicating that there was some reason that one object was placed above another one. An example of one[5] pictorial representation of sentence 56 is shown in figure 6.15.

(56) Anna saw the cat on the table.

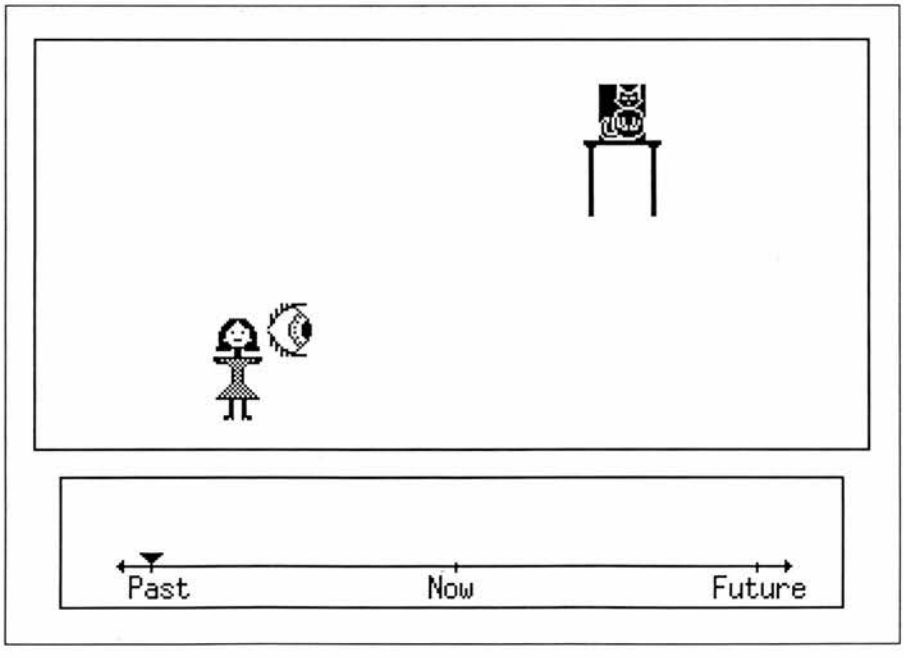


Figure 6.15: Picture, with Gravity not implemented

[5] There are *two* logical form representations of this sentence. The one chosen is the one where the 'cat' is on the 'table', rather than the seeing event taking place on the table.

6.6 Herskovits' Elementary Spatial Concepts

The system used in this thesis uses seventeen elementary spatial concepts to pictorially represent the *meaning* intended by the spatial expressions. These spatial concepts are all motivated by representation via a picture [Pylyshyn 73], [Mani & Johnson-Laird 82], [Johnson-Laird 83], [Garnham 87].

Annette Herskovits describes an elementary set of spatial concepts in her book, "Language and Spatial Cognition." I will show that my system is compatible with her concepts. Further, I believe that my set of spatial concepts is a more general description of her set. I will redefine some of her concepts into my set of pictorial primitives representing spatial expressions. The reason my set is smaller is because most of the concepts are simplified when described in pictorial descriptions.

Herskovits presents a list of elementary spatial concepts that occur in the *ideal* meanings of prepositions. She proposes five classes of concepts:

- (1) topological,
- (2) geometric,
- (3) physical,
- (4) projective, and
- (5) metric.

In each of the *classes*, they may have attributes, properties, objects, or relations, which she associates with ideal meanings of prepositions.

A sample of some of the elementary spatial concepts she proposes [Herskovits 86, page 55] are listed below:

```
(1)topological:
   relations
       enclosure (in, inside)
       contiguity with line or surface (on, against)
```

```
        order of three points on a line (between, beyond)
        order of two points on an oriented line (above, under,...)
        line in/on a plane (across)

(2) geometrical:
   relations
        alignment of points (between, behind, ...)
        parallelism of lines (along)
        alignment with direction (e.g. vertical) (over, under,...)
        orthogonality of lines (to the right/left ...)

(3) physical:
   objects
        vertical direction (above, below, ...)

(4) projective:
     relations
          on line of sight (behind,in front of ...)
          on orthogonal to line of sight (to the right)

(5) metric:
    attributes
          distance (near, close to, ...)
```

I will discuss the *topological* and *geometrical* concepts after first discussing her next three spatial concepts.

The concepts she listed that are *physical* are above, below, left, right, et cetera. I believe that several of these are actual projective spatial expressions or deictic expressions. These deictic expressions are simplified because the pictorial representation builds the picture with the viewpoint from the speaker, thus making the expressions have the appropriate meaning to the viewer. This works in two-dimensional space as long as the viewpoint

is not moved, which could be the case if discourse problems of multiple sentences were introduced.

The *projective* relations will definitely depend upon the viewpoint. Again, the problem is simplified as the viewpoint is fixed to a chosen location that shows the relations with respect to a viewer looking directly at the objects being described. "The man saw the cat to the left of the table" was shown in the previous section. Other viewpoints could be chosen, but they do not yield as much as setting the viewpoint to that of the speaker. Pictorially the problem of physical direction and projective are identical.

She defined the *metric attributes* as a list of all of the prepositions that fill this category. My pictorial primitives make use of the set of *touching*, *near*, *far-from* (and their not operations), which all can specify the distance attribute. A pictorial representation would show the two following sentences with the same pictorial representation as the two spatial prepositions are defined identically in terms of pictorial primitives.

(57) The bike is near the house.

(58) The bike is close to the house.

Of course, a user of the system can redefine a preposition to capture some difference if so desired. Most important is that these types of attributes are vague. Herskovits does not offer a working system or a methodology of how to build a working system from her concepts. She describes that there is a difference between near and far, but no mechanism is provided to actually represent it or to implement it. In the Text-To-Pictures System the vague concepts of *near* and *far-from* relations are defined in terms of percentages of the total viewing window. If the total viewing window is 200 pixels length in the vertical, and if *far-from* is defined as 60% of the vertical distance, then by using constraint satisfaction problem technique one object can constrained to be over 120 pixels away. The user can define these, however after some use I found that 5% for near, and 60% for *far-from* seemed to work well and be understood adequately by the viewers who used the system.

For her first two concepts, *topological* and *geometrical*, several of the prepositions she described (*on*, *in*, *beside*, *to the left of*), were all represented pictorially using the set of spatial primitives. For example, placing two *objects*, or two *points* next to each other is the

same in pictorial terms. The example of *in* shows, that combining the spatial primitives with naive physics to generate a picture, simplifies the problem of worrying about whether an object can protrude from another, so it is in only partially in. Although, the Toulouse group distinguished two types of *in* and *dans* preposition (partially in and fully within), it was handled by the *same* mechanism of spatial primitives and naive physics.

If a definition needs to be more restrictive it can be easily be made to be so. Herskovits describes two objects that are between each other (geometrical) and also discusses three points in a line, with one of them as in between (topological). The spatial primitive *between* defines a region where the epicenter of the object must be contained within the region of the corners of the other two objects. This definition means that points, objects, even objects that are so large, that they might overlap another object could be constrained into that region. However, if some specific definition of *between* is being used, such as a special domain in mathematics, where the two points are along a line and the third point is in the middle, *nothing* would need to be changed in my pictorial definition. The "between region" between two points in a line, will also be a line, and the epicenter of the third object must be on that line. If the third object is a point then it fits exactly the case of three points on a line. However, my definition is more general in that an *object* could between two *points*.

In summary, using a pictorial definition simplifies the spatial expression representation problem. Using spatial primitives combined with naive physics give a powerful mechanism for representing spatial expressions. Finally, this technique is an improvement over the work of Vandeloise, the Toulouse Group, and Herskovits, in that this framework of representing spatial expressions as a constraint satisfaction problem allows a working system to be developed.

Chapter 7

Temporal Expressions

This chapter shows pictorial representations of temporal expressions. Temporal expressions relate objects or events with time. However, the way they relate time can be of many different forms. For example, the time of an event may be absolutely referenced or relatively referenced. Event times that are relatively referenced may have exact start and end times, or they may be vague. In this chapter, example sentences containing various temporal expressions will be presented with their associated pictorial representation. First, the set of pictorial primitives that I use to represent temporal expressions is presented.

7.1 Pictorial Primitives for Temporal Expressions

There are several knowledge sources that contain constraints for the pictorial generation process. At the lowest level, these constraints are defined in terms of a set of pictorial primitives. Examples of such primitives for Temporal Expressions are: before, after, during, et cetera. These examples are two-place predicates that describe the relationship between an event and time.

Considering an event E along with T_1 and T_2 representing time, the set of Temporal Expression Pictorial Primitives is:

- Interrelationships 1 and 2 are two-place relations with some event and time. The beginning or the end of the event is known.
- Interrelationships 3 and 4 are two-place relations with some event and time. The

	primitive	*description*
1	begin(E,T)	Event E started at time T
2	end(E,T)	Event E ended at time T
3	after(E,T)	Event E occurred sometime after time T
4	before(E,T)	Event E occurred sometime before time T
5	duration(E,T_2)	Event E lasted for duration T_2

exact time of the event is not known.

- Interrelationship 5 is a two-place relationship with a different type of time. The duration of the event was T_2.

The five temporal primitives have an inter–relationship among themselves. Intuitively, the duration is the difference in time from the end of an event and the beginning. Also, given a time T_1 that applies to *begin(*E, T_1*)*, would have to be more precise than a time T_2 that applies to the primitive *after(*E, T_2*)* for the same event E.

For some event E, and given that:

begin(E, T_1*)*

after(E, T_2*)*

Then the following is true:

$$T_1 > T_2$$

A time relation relates these five functions. The **time relation** is defined as:

time-of-event(Begin,End,After,Before,Duration)

- Begin–exact time of the *beginning* of the event.
- End–exact time of the *completion* of the event.
- After–only know that the event *started after* time x.
- Before–only know that the event *finished before* time x.
- Duration–length of time of event. It should equal *End* minus *Begin*.

With this structure one can define temporal expressions. This representation of time is a simplied scheme based directly from Allen's interval algebra [Allen 83]. Many representation schemes have been proposed for temporal representation; Allen's algebra of temporal intervals is one of the most popular.

Given this temporal representation, a pictorial representation of the temporal representation is presented. Allen's primitives were illustrated pictorially in [Vilian *et al* 86], showing time increasing from left to right, and marking the beginning and end of events, and showing duration. My pictorial representation is an extension of that illustration method.

The pictorial representations of all of the permutations of the time relation are shown in figure 7.1. The first pictorial representation is when a definite *begin* and *end* are known. The dark circle indicates that it is a definite beginning or ending. Where the open circle indicates the event occurred *before* or *after* that time. This is analogous to the less-than-or-equal-to and less-than relations and how they are typically shown on a number line. A dark line indicates that the event definitely took place for a duration of time that is shown. The first example shows a definite begin and end (marked by closed circles) and a solid line showing the duration of the event. The fourth example, "*Begin Only*," has a definite beginning marked by a dark circle, but the duration is not known. The ending time is not known, thus the dashed line continues with an arrow. Four examples are shown that have a definite duration of time (and is shown by a solid line) but the exact begin and end are not known. The final example of (*begin* and *after*) and (*end* and *before*) are not allowed because they are contradictory.

The temporal representation technique is shown in a Temporal Pictorial Representation Window. In the next three sections, these temporal expressions are defined in terms of the time relation to provide the correct primitives to generate a pictorial representation in the Temporal Pictorial Representation Window.

Given the definitions of the temporal pictorial primitives, examples of absolute reference, relative reference, and duration (which are illustrated by the temporal prepositions *at*, *after*, and *for*, respectively) are presented.

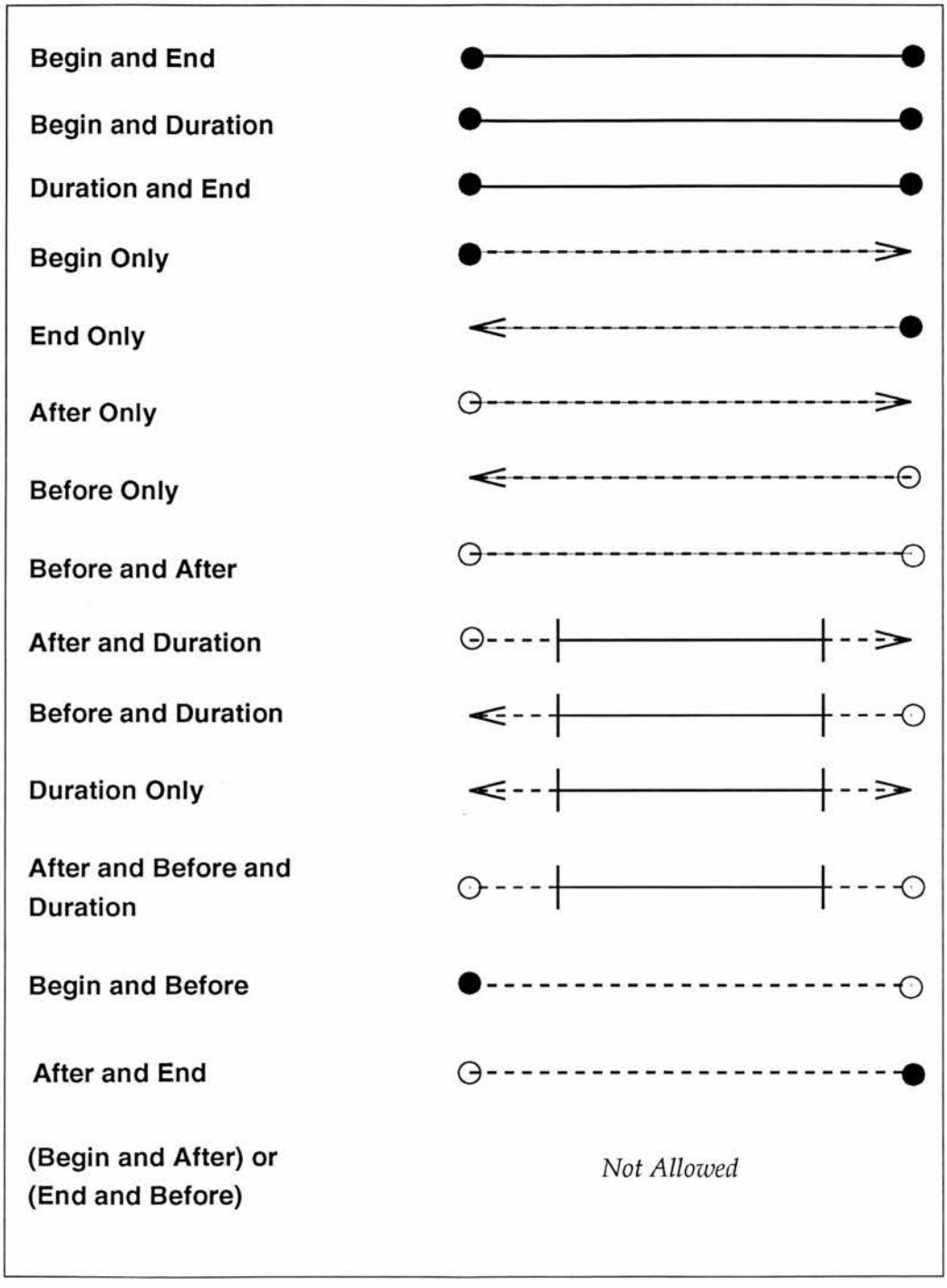


Figure 7.1: Pictorial Representations of the Time Relation

7.2 Absolute Reference

Absolute time reference has an exact beginning or an end to an event. Examples of such temporal prepositions that have absolute time references are *at* and *until.*

(59) There is a meeting at three o'clock.

(60) Suresh drove the car until six o'clock.

In sentence 59, the beginning of the event is known. This is marked by the *at* temporal preposition. The end of an event can also be determined, as in sentence 60.

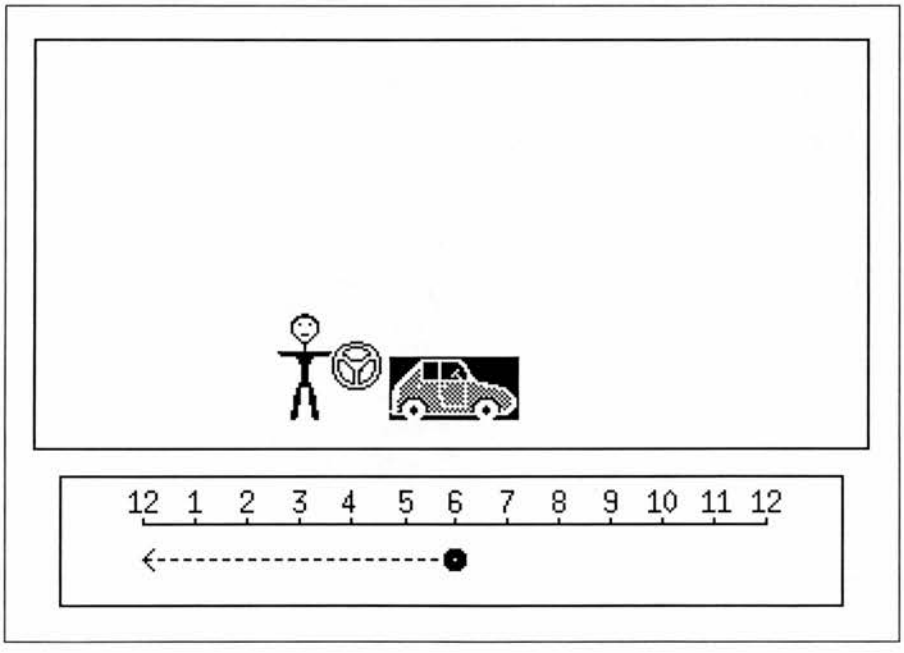


Figure 7.2: PR of *until*

The temporal PRW for sentence 60 is shown in figure 7.2.

The two prepositions are defined via the time relation. The time relation again is:

time-of-event(Begin,End,After,Before,Duration)

and the two prepositions are defined as:

$at(T_1) \Rightarrow$ *time-of-event*$(T_1, ?, ?, ?, ?)$

$until(T_2) \Rightarrow$ *time-of-event*$(?, T_2, ?, ?, ?)$

Notice that the event is of some unknown duration that completes at 6 o'clock. The dotted line shows that event is not exactly that it occurred at the time show in the chart. Although, the dashed line is shown, for example, above 3 o'clock, there is no information of whether the event was taking place at that time or not.

7.3 Relative Reference

Times that indicate the event occurred before or after sometime, but do not give definite begin and end times are referenced relatively.

(61) There is a meeting before Saturday.

(62) Claire drove the car after two o'clock.

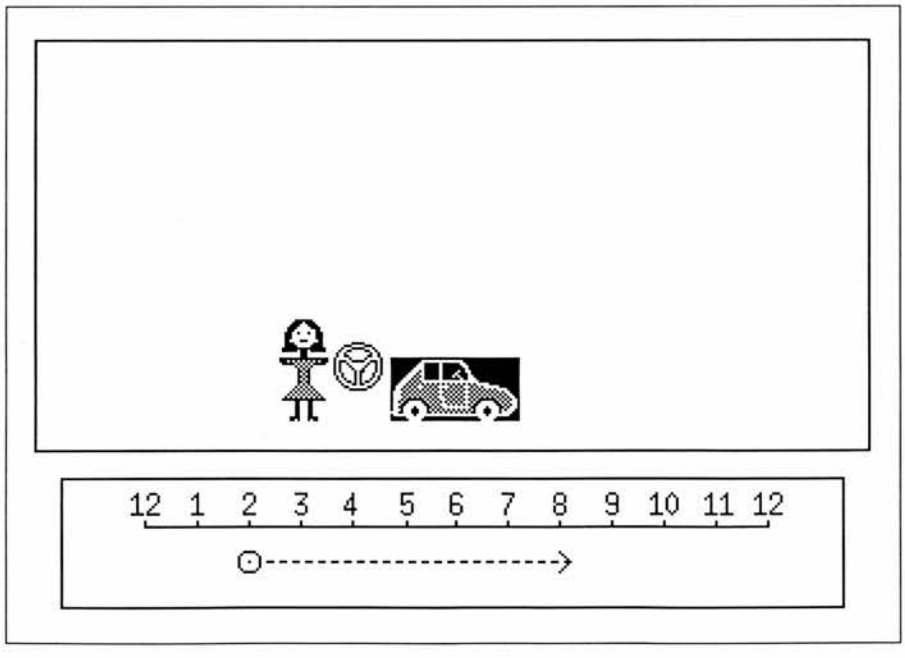


Figure 7.3: PR of *after*

The PRW time of event window for sentence 62 is shown in figure 7.3.

Given the time relation, the temporal expressions *after* and *before* can be defined via the time function as:

$$after(T) \Rightarrow time\text{-}of\text{-}event(?,?,T,?,?)$$
$$before(T) \Rightarrow time\text{-}of\text{-}event(?,?,?,T,?)$$

In this example, the temporal representation uses an open dot to represent that the event did not start exactly at 2 o'clock, rather it started after 2 o'clock. Also, in this example, there is no indication of how long an event lasted. Duration is another component of time relation and is discussed next.

7.4 Duration of Event

Often the *duration* of an event is specified. It may or not also contain references to the begin or end of an event. An example of duration being specified is the use of the *for* preposition, as in shown in sentence 63.

(63) Andy drove the car for three hours.

The duration must be positive, and is shown in figure 7.4. The pictorial representation uses the following scale with zero as the minimum duration. The scale is shown in hours.

Duration is described with the time-of-event function as:

$$for(T) \Rightarrow time\text{-}of\text{-}event(?,?,?,?,T)$$

7.5 Non–specific Time Reference

The following three sentences all include non–specific time reference. The example includes a sentence that has absolute reference, relative reference, and one of duration.

(64) Sheila left on Ramajadeen. [absolute reference]

(65) Alison slapped him after his comment. [relative reference]

(66) Pete played during the game. [duration]

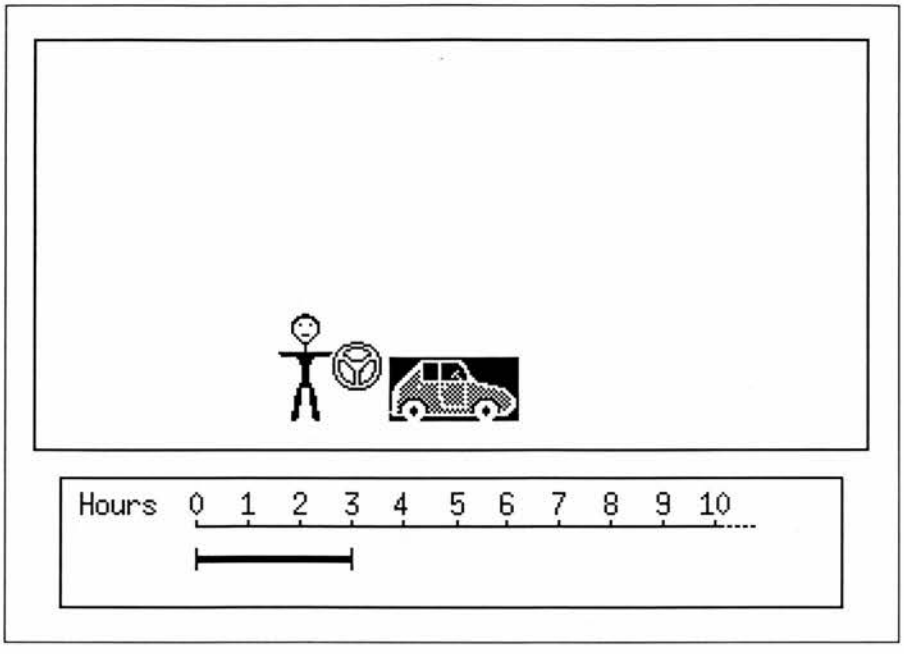


Figure 7.4: PR of *for*

All three of these sentences, which contain **non–specific time reference**, are referencing the time of some event to another event, as opposed to some specific time. To solve this problem, one could approach the problem as a translation process to a known time system (i.e. Ramajadeen is on 24 March), thus converting the time to an absolute time reference. This technique is used in the Core Language Engine. All dates are referenced to a specific date. The following is an example of how temporal information is given in the CLE logical form:

[precedes_in_time,E,SF(date(1991,12,6))]],

The date in the logical form interpreted as the 6th of December 1991. *E* represents the event. The entire phrase is saying that some event preceded 6 December 1991. All dates are converted to this format in the CLE system. Sometimes this is a problem, as is shown with the following sentence 67.

(67) Alan said he saw her on Tuesday.

The Core Language Engine translates Tuesday to a date. Since past tense was marked, the CLE looks at the current date that the sentence is being processed, and assumes

the date of "Tuesday" to be the most recent Tuesday. This could lead to problems, and causes the pictorial representation to display information that is not intended. I modified the CLE representation for common time scales, such as day of the week, so that pictorial representation will shown "Tuesday," if that was the time referenced in the sentence, rather than a specific date.

Another problem is that, sometimes, the translation process may not be able to be accomplished. There may not be any more information about the other event. The only time relation mentioned may be relating the two events, although there is no idea of actually when either event specifically occurred.

7.6 Determining Time Scale

Determining the *scale* of time is a problem that is quite important for pictorial representation. The problem is that the text-to-pictures system has to choose some time scale to plot the events against. There are several approaches.

One approach is to simply use the time scale of the highest granularity for all of the events. However, this has the problem that some scales could not be shown. Look at the following examples:

(68) Sue played badminton since 4 o'clock.

(69) Jussi played badminton since Saturday.

(70) Rob played badminton since June.

(71) Suresh played badminton since 1966.

In sentence 68, the event is shown on a different time scale, than in sentence 69. In fact all of the four sentences each need a different scale. It would be difficult to shown sentence 71 on the first scale (hourly scale) used in figure 7.5.

In the Text-To-Pictures System, the scale used is the one that is stated. If a date is specified a scale showing all the days of the month, with an arrow pointing to the day. If a day of the the week is used, then the "day of the week" scale is shown, with a pointer

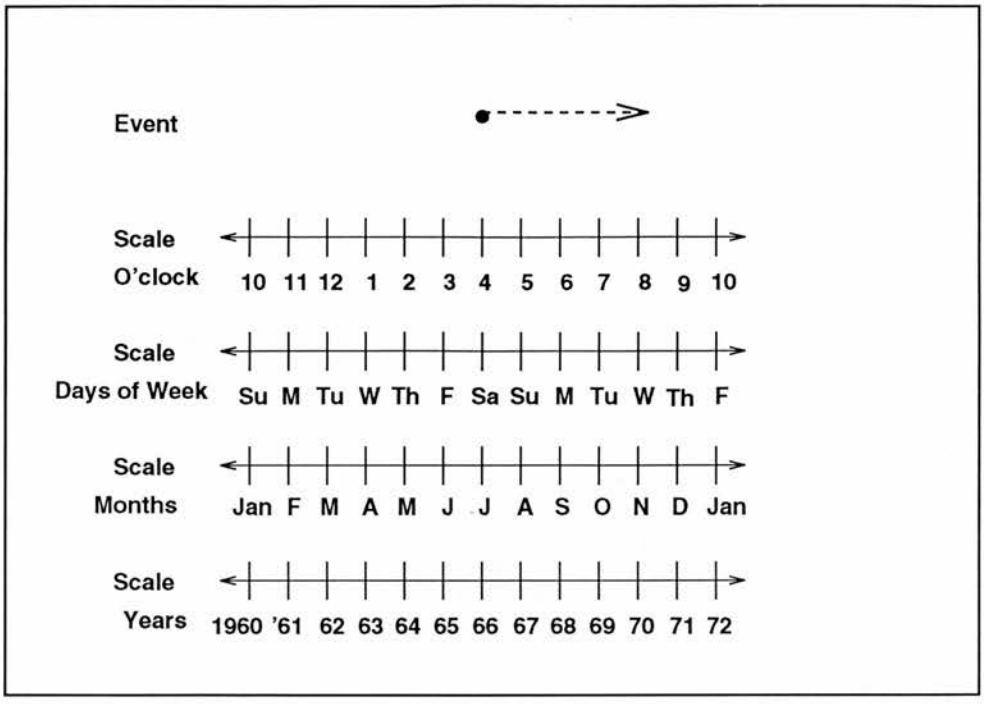


Figure 7.5: Varying Time Scales

displaying what day the event occurred on. If the time is simply in the "past" then the past is marked on a "past–now–future" scale.

(72) Alan said Ian saw the woman on Tuesday.

A pictorial representation of sentence 72 is shown in figure 7.6, which shows both the day of the week and the past tense being displayed.

Another example of varying time references is given in sentence 73.

(73) After driving for two weeks, the car exploded in less than a second.

Sentence 73 does not cause a problem as the first temporal expression "after driving for two weeks..." would be shown in the temporal PRW corresponding the first spatial PRW. The second temporal expression, "in less than a second," shown in another temporal PRW for the event of the explosion.

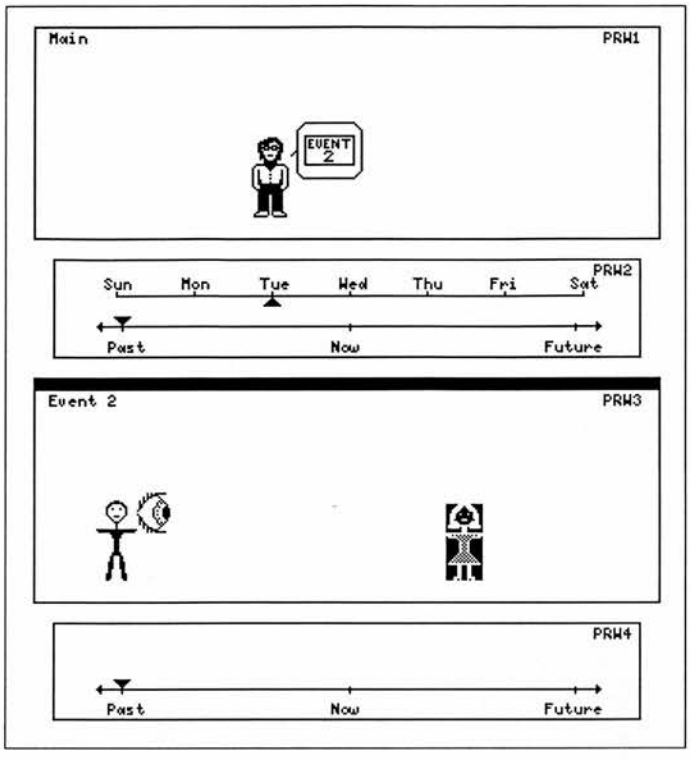


Figure 7.6: Two Different Temporal Scales in One PR

7.7 Temporal Negation

Negation of the verb was shown in section 5.7, where reverse video of the spatial pictorial representation window indicated the "negation" of the event shown. This technique also works when a temporal expression is involved.

(74) Alan said that Ian didn't see the woman on Tuesday.

For example, if the system is given sentence sentence 74, that contains a negation and a temporal expression, the following pictorial representation is given in figure 7.7. The pictorial representation shows in the spatial pictorial representation window that the event did *not* happen. While in the temporal pictorial representation window, it shows *when that event did not happen.*

A specialized technique of representing temporal expressions should be addressed, because upon initial inspection it seems to work. This technique was not implemented in this thesis, however, because it would misrepresent what was intended for many temporal

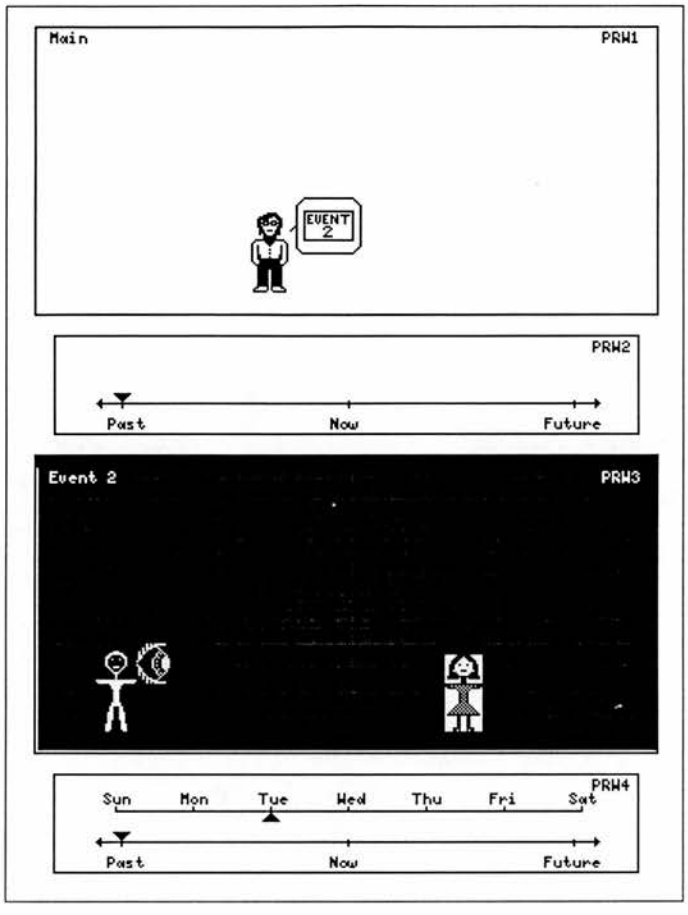


Figure 7.7: PR of Negation and Temporal Expression

expressions. The technique could be used for negation of temporal prepositions, such as "since", by modifying both the spatial and temporal PRWs, in attempt to make both windows show when the event did happen, as opposed to when it did not happen.

(75) Alan did not drive the car since 6 o'clock.

The implemented system represents sentence 75 with the spatial PRW negated (reverse video), and the temporal window showing since 6 o'clock. Using the special technique for "since", the spatial window would be shown without the negation (without the reverse video) and the time arrow would be "flipped" to show the event occurred "until" 6 o'clock. The problem is that this technique only works for a few cases. It does not work for temporal points, such as "at" or "on". It makes no sense, at all, to negate durations in that manner, and finally, the reversing technique could yield false results, as shown in

the sentences 76 and 77.

(76) Alan did not drive the car after Tuesday.

(77) Alan did drive the car before Tuesday.

Sentences 76 and 77 are not equivalent statements. For the several reasons stated, this specialized technique was not adopted. Therefore, negation of temporal expressions is handled the same way that other expressions are–by negating the event in the spatial PRW.

Chapter 8

Ambiguity and Vagueness

This chapter looks at the type of problems that occur in both textual representations and in pictorial representations: ambiguity and vagueness.

The first section discusses the problems with ambiguity, and shows an example of how the working system in this thesis is used to display several interpretations of an ambiguous sentence. A user could then choose the correct or intended interpretation of the sentence by pointing the mouse to the appropriate pictorial representation of the sentence.

The second section discusses the problem of vagueness. A picture usually requires specific information for it to be drawn. An object must be drawn in some exact location, and a specific icon must be used to represent an object. A method to determine useful scaling of information, the missing–information module, and the use of generality in absence of specificity, are all described as techniques to reduce the problems caused by vagueness.

8.1 Ambiguity

Ambiguity was previously discussed in section 3.3.2. In that section, four types of ambiguity were identified:

1. Lexical Ambiguity
2. Syntactic Ambiguity
3. Semantic Ambiguity
4. Pictorial Ambiguity

Using the Text–to-Pictures System in this thesis, most textual lexical ambiguity can be resolved. An example of lexical ambiguity within a similar category, is given in sentence 78.

(78) He sat near the bank.

In the example sentence, the noun "bank" could mean a bank with money or a bank next to the river. Each would have a different icon representation. This would also mean each would have a different pictorial representation. Two logical forms are generated for this sentence as there are two occurrences of the word "bank" in the lexicon. One representation is *bank.ContainsMoney* and another is *bank.RiverEdge*.

Both syntactic ambiguity and semantic ambiguity are shown in an example in the next section, where the Text-To-Pictures System displays all of the possible interpretations generated for a sentence that is both syntactically and semantically ambiguous.

The fourth kind of ambiguity, pictorial ambiguity, is an unwanted effect of pictorial representation. Minimal pictorial ambiguity is a goal of a good pictorial representation. Pictorial representation can remove or reduce the other three types of ambiguity, however, pictorial ambiguity can be introduced.

The primary cause of pictorial ambiguity is that the pictorial language is not clearly defined to the viewer. Chapter 5, "General Linguistic Expressions," defined the general interpretation of the pictorial representation. In addition to the general method of interpreting a pictorial representation, the icons are defined with an **icon definition window**, which is displayed before the several pictorial representation windows of a sentence. The icon definition window shows each of the icons and the textual word they represent. An example of a icon definition window is shown in figure 8.1.

This defines the icon that shows the man, and gives an icon for the woman as "Carla." It defines the "seeing event", a "telescope", and a telescope being used as an instrument. It also defines a "hill." The viewer may have his or her own mental image of a hill, but this defines that for the following pictorial representations the icon shown above the word "hill" will be used. The same is true for each of the icons.

Also shown in figure 8.1 is the number of logical forms (LFs). There are seven for that

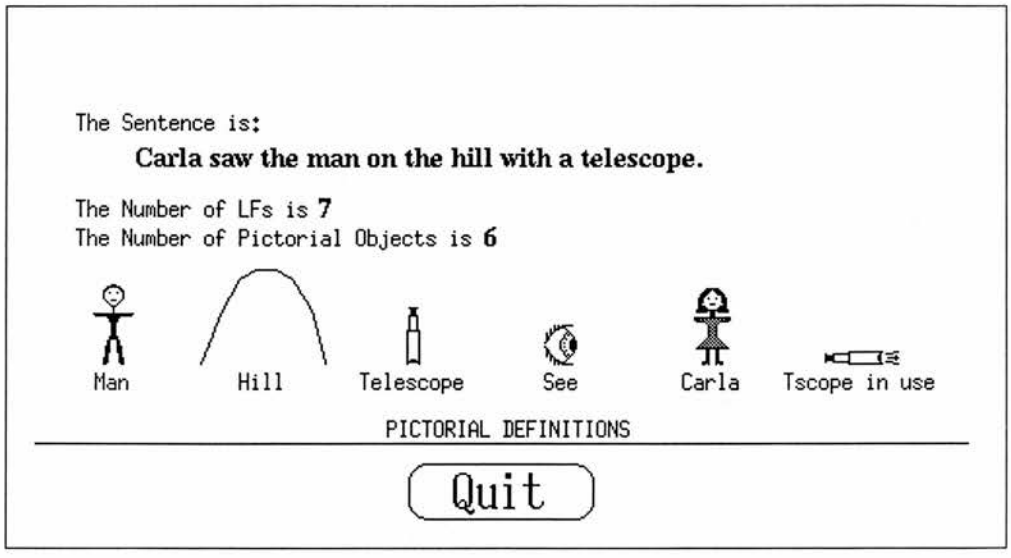


Figure 8.1: Icon Definition Window

ambiguous sentence, which will be fully discussed in the next section.

8.1.1 A System to Disambiguate Sentences

An interesting technique to identify the intended meaning of an ambiguous sentence, is to pictorially display all of the possible meanings of the sentence, and let the reader or viewer choose the correct representation. Often it is difficult to think of all of the possible meanings of an ambiguous sentence without constructing a quick sketch of the sentence. An instructor of a computational linguistics course once gave as an example an ambiguous sentence, and then drew a simple cartoon–like drawing to show the differences among each of possible meanings of the sentence [Pain 89]. The system used in this thesis can be used to represent all the logical forms generated for a particular sentence. Sometimes, some of the possible logical forms are not plausible. It may be hard, to judge the plausibility of a certain reading, to determine all of the possible meanings, or even grasp all of the operations described within one logical form, without first drawing a picture. The following is an example of such a system.

Two types of ambiguous sentences can be displayed using the Text–To-Pictures System. Here are examples of the two types:

(79) Carla saw the man on the hill with a telescope.

(80) Alan said Ian saw the woman on Tuesday.

The first example is a spatial ambiguity example representing sentence 79. Sentence 79 is also semantically ambiguous due to the two senses of the 'with'. There are seven different interpretations of sentence 79. The seven different logical forms will be shown, as well as the conversion process, and the resulting seven pictorial representations for that sentence. The other example sentence 80 is temporally ambiguous. There are two logical forms generated for that sentence, and the resulting pictorial representations are given immediately after the spatial ambiguity example.

8.1.2 Spatial Ambiguity Example

1st LF for Spatial Ambiguity Example

Here is the logical form description for the first interpretation of sentence 79. The reading of this logical form is that is the hill has a telescope, the man is on the hill, and Carla sees that man.

```
%  1st Logical Form of:
% "{Carla} saw {{the man} on {{the hill} with {a telescope}}}."
[dcl,
 quant(exists,
       A,
       [and,
        [man_MalePerson,A],
        quant(exists,
              B,
              [and,
               [hill_HighGround,B],
               quant(exists,
                     C,
                     [telescope_Device,C],
                     [with_Accompanying,B,C])],
              [on_Locational,A,B])],
       quant(exists,
             [and,
              [event,D],
              quant(exists,E,[current_time,E],[precedes_in_time,D,E])],
             [see_LookAt,D,carla,A]))]
```

One temporal constraint was created from the logical form. The seeing event "preceded in time" the current time, so the temporal window for that event is constrained to past

tense.

```
Temporal Constraints are:

   plot_time_and_scale(temporal, past, past_now_future)
```

Several spatial constraints were generated from the logical form. The 'see' verb, the 'on' and 'with' prepositions each created several constraints. The interpretations of these relations for this reading are:

- See – to look at (as opposed to realize).
- On – on a location (as opposed to time).
- With – with accompanying or having (as opposed to an instrument).

The 'see' (see_LookAt) relationship generates the following pictorial constraints:

$see(Event, Actor, Object) \Longrightarrow$
left(Actor,Event),
touching(Actor,Event),
in_top_third(Event,Actor),
left(Event,Object),
farfrom(Event,Object).

The 'on' (on_Locational) relationship generates the following pictorial constraints:

$on(A, B) \Longrightarrow$
touching(A,B),
above(A,B),
not(left(A,B)),
not(right(A,B)),
in_middle_horiz(A,B).

Finally, the 'with' (with_Accompanying) relationship generates the following constraints:

$with(A, B) \Longrightarrow$
touching(B,A),
left(B,A),
not(above(B,A)),
not(below(B,A)).

Therefore, the total spatial constraint list generated from the logical form, listed in order of priority, is:

```
Spatial constraints are:

   touching(carla, see_LookAt, 9)
   farfrom(see_LookAt, man_MalePerson, 9)
   touching(man_MalePerson, hill_HighGround, 9)
   left(carla, see_LookAt, 8)
   left(see_LookAt, man_MalePerson, 8)
   above(man_MalePerson, hill_HighGround, 8)
   touching(telescope_Device, hill_HighGround, 8)
   not left(man_MalePerson, hill_HighGround, 7)
   not right(man_MalePerson, hill_HighGround, 7)
   not above(telescope_Device, hill_HighGround, 7)
   not below(telescope_Device, hill_HighGround, 7)
   left(telescope_Device, hill_HighGround, 7)
   in_top_third(see_LookAt, carla, 5)
   in_middle_horiz(man_MalePerson, hill_HighGround, 5)
```

The number listed at the right of each constraint is the associated priority between levels one and ten. Using the definition of gravity in the naive physics module, which was described in section 6.5, the gravity constraint is tested against each of the five pictorial objects, to see if it should apply. It only has an effect on 'Carla', the 'hill', and the 'telescope'.

```
Applying Naive Physics...

Try to apply Gravity to object carla
Bottom Variable is $_22812  [0..100]
Gravity has an Effect!

Try to apply Gravity to object see_LookAt
Bottom Variable is $_22908  [17..30]
Gravity does NOT effect object see_LookAt

Try to apply Gravity to object telescope_Device
Bottom Variable is $_23004  [0..98]
Gravity has an Effect!

Try to apply Gravity to object hill_HighGround
Bottom Variable is $_23100  [0..30]
Gravity has an Effect!

Try to apply Gravity to object man_MalePerson_Obj
Bottom Variable is 50
Gravity does NOT effect object man_MalePerson_Obj

Applying Nice Pictures...
Applying Missing Information...
```

After the constraints are generated from the logical form, the CSP solver is activated. Each constraint can reduce the range of possible legal solutions of each variable. The naive–physics, nice–pictures, and missing–information modules work interactively with CSP solver. If a further constraint is needed it is applied. At the end, the CSP Solver gives the following solution[1].

Notice that gravity did not further constrain the icon representing the *man_MalePerson*. This is because the *man_MalePerson* is already constrained to be above the hill. Therefore, it is impossible for the *man_MalePerson* to touch the ground. This is evident by looking at the range of acceptable *bottom* variables (the variable that describes the bottom of an icon). For the *hill_HighGround* the bottom variable can be any value between 0 to 30. When gravity is applied, the hill's bottom value is constrained to touch the ground

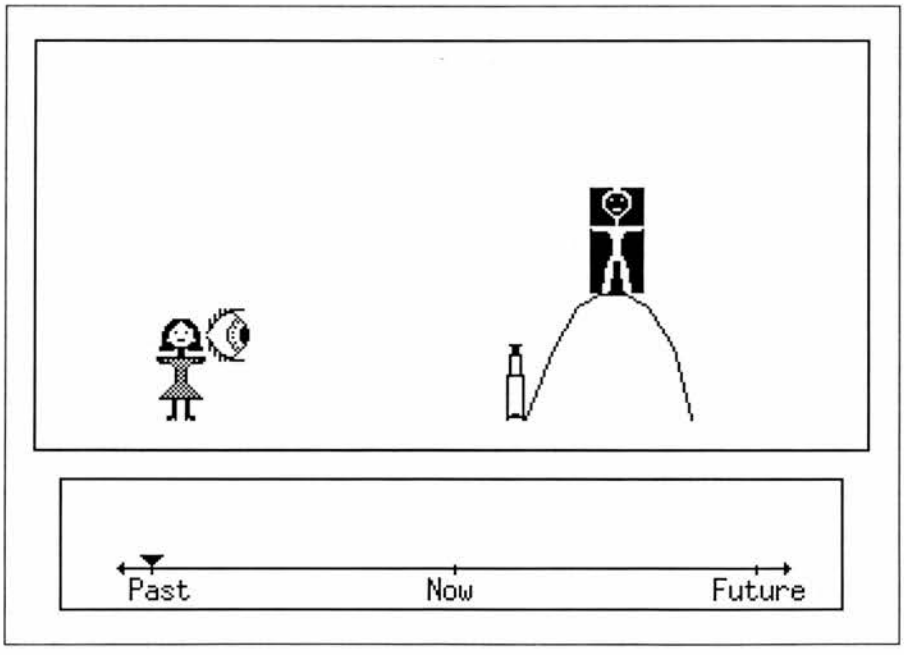
[1] The first component of the object record is the object tag, the second is the word the pictorial object represents, the the left side of the icon (low x value), followed by the right, bottom, and the top sides of the icon.

by setting it to 0. The *man_MalePerson* bottom variable is already set to 50. Therefore, it can't touch the ground. Thus, gravity does not further constrain the icon representing the man.

```
CSP SOLUTION is:
pictorial_object([obj5, carla, 35, 54, 0, 40])
pictorial_object([obj4, see_LookAt, 54, 73, 23, 44])
pictorial_object([obj3, machine_Device, 170, 177, 0, 30])
pictorial_object([obj2, hill_HighGround, 177, 242, 0, 50])
pictorial_object([obj1, man_MalePerson_Obj, 202, 223, 50, 92])

plot_time_and_scale(temporal, past, past_now_future)
```

This is picture for the generated solution, or the pictorial representation of the first interpretation of the sentence.



Pictorial Representation for Logical Form One

Notice that Carla is looking at the man. The man in is in reverse video to indicate that he is the object of the viewing (as opposed to the hill or the telescope). The man is on the hill, and not touching the telescope to imply any relation between the man and the telescope.

2nd LF for Spatial Ambiguity Example

Here is the logical form description for the 2nd interpretation of the sentence. This interpretation of the reading, is that Carla sees the man, but Carla is on the hill. The telescope is accompanying the hill, and is in not in Carla's possession.

```
%  2nd Logical Form of:
% "{Carla} saw {the man} on {{the hill} with {a telescope}}."
[dcl,
 quant(exists,
       A,
       [man_MalePerson,A],
       quant(exists,
             B,
             [and,
               [event,B],
               quant(exists,
                     C,
                     [current_time,C],
                     [precedes_in_time,B,C])],
             [and,
              [see_LookAt,B,carla,A],
              quant(exists,
                    E,
                    [and,
                     [hill_HighGround,E],
                     quant(exists,
                           E,
                           [telescope_Device,E],
                           [with_Accompanying,D,E])],
                    [on_Locational,B,D])]))]
```

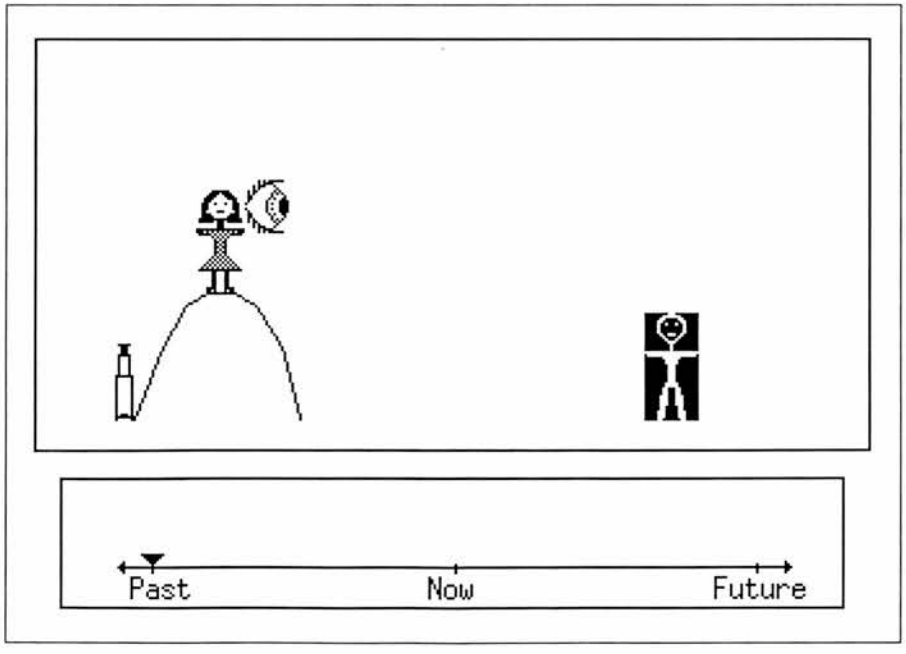
The only difference, between interpretation of logical forms number 1 and number 2, is that 'Carla' is on the hill rather than the 'man' being on the hill[2]. The same relations (*see.LookAt* and *on_Locational*) are used as were in logical form number 1, however, with the relations acting upon different objects. The set of spatial constraints is given in the next box.

[2] The specific location of the man is not given in the sentence, therefore in the picture, the man is located by himself to prevent false implicatures, although it is possible that the man could also be on the hill with Carla.

Again, the temporal information is extracted from the logical form. The seeing event "preceded in time" the current time, so the temporal window for that event is constrained to past tense. This sentence is not ambiguous temporally, so all of the logical form descriptions in this example will contain the same temporal information. For the rest of the spatial ambiguity examples, I will not show the temporal information.

```
Spatial constraints are:
   touching(carla, see_LookAt, 9)
   farfrom(see_LookAt, man_MalePerson, 9)
   touching(carla, hill_HighGround, 9)
   left(carla, see_LookAt, 8)
   above(carla, hill_HighGround, 8)
   left(see_LookAt, man_MalePerson, 8)
   touching(telescope_Device, hill_HighGround, 8)
   not left(carla, hill_HighGround, 7)
   not right(carla, hill_HighGround, 7)
   not above(telescope_Device, hill_HighGround, 7)
   not below(telescope_Device, hill_HighGround, 7)
   left(telescope_Device, hill_HighGround, 7)
   in_top_third(see_LookAt, carla, 5)
   in_middle_horiz(carla, hill_HighGround, 5)
```

Graphing the solution gives the following pictorial representation:



Pictorial Representation for Logical Form Two

3rd LF for Spatial Ambiguity Example

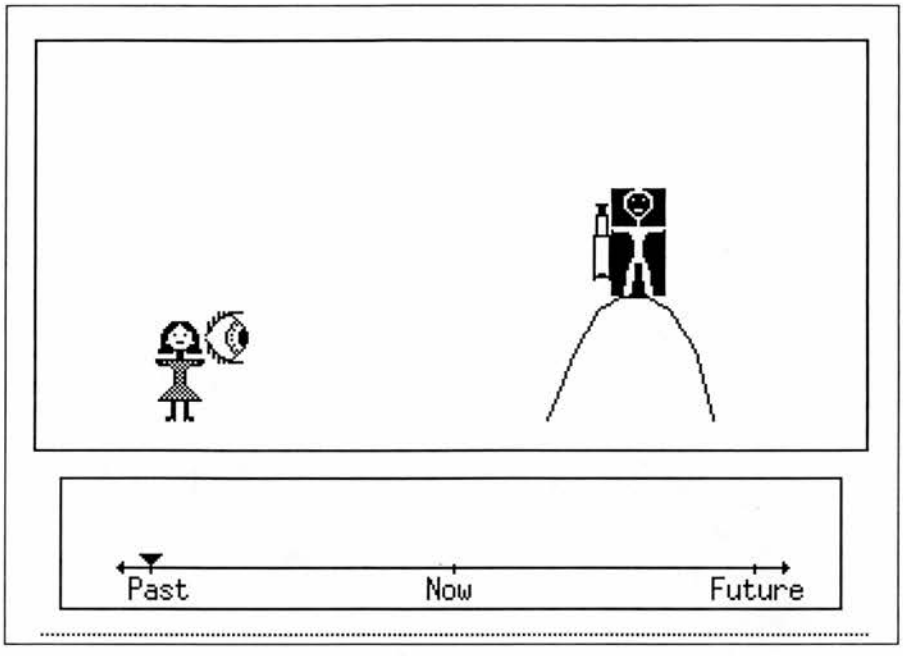
The third logical form is the interpretation of the sentence, that Carla sees a man, the man is on the hill, and the man has the telescope. This differs from reading 1, in that the telescope is not accompanying the hill. The logical form for reading 3 is:

```
%  3rd Logical Form of:
% "{Carla} saw {{{the man} on {the hill}} with {a telescope}}."
[dcl,
 quant(exists,
       A,
       [and,
         [and,
           [man_MalePerson,A],
           quant(exists,B,[hill_HighGround,B],[on_Locational,A,B])],
         quant(exists,C,[telescope_Device,C],[with_Accompanying,A,C])],
       quant(exists,
             D,
             [and,
               [event,D],
               quant(exists,E,[current_time,E],[precedes_in_time,D,E])],
             [see_LookAt,D,carla,A]))]
```

The set of spatial constraints generated to represent this logical form are:

```
Spatial constraints are:
   touching(carla, see_LookAt, 9)
   farfrom(see_LookAt, man_MalePerson, 9)
   left(carla, see_LookAt, 8)
   touching(man_MalePerson, hill_HighGround, 8)
   above(man_MalePerson, hill_HighGround, 8)
   left(see_LookAt, man_MalePerson, 8)
   touching(telescope_Device, man_MalePerson, 8)
   not above(telescope_Device, man_MalePerson, 7)
   not below(telescope_Device, man_MalePerson, 7)
   left(telescope_Device, man_MalePerson, 7)
   not left(man_MalePerson, hill_HighGround, 7)
   not right(man_MalePerson, hill_HighGround, 7)
   in_top_third(see_LookAt, carla, 5)
   in_middle_horiz(man_MalePerson, hill_HighGround, 5)
```

The pictorial representation for logical form 3 is:



Pictorial Representation for Logical Form Three

4th LF for Spatial Ambiguity Example

This is the interpretation that the man is on the hill, and that Carla is holding (accompanied by) the telescope, while she sees the man without using the telescope. This arises because of the lexical ambiguity associated with the preposition *with*.

```
%  4th Logical Form of:
% "{Carla} saw {{the man} on {the hill}} with {a telescope}."
[dcl,
 quant(exists,
       A,
       [and,
        [man_MalePerson,A],
        quant(exists,B,[hill_HighGround,B],[on_Locational,A,B])],
       quant(exists,
             C,
             [and,
               [event,C],
               quant(exists,D,[current_time,D],[precedes_in_time,C,D])],
             [and,
              [see_LookAt,C,carla,A],
              quant(exists,
```

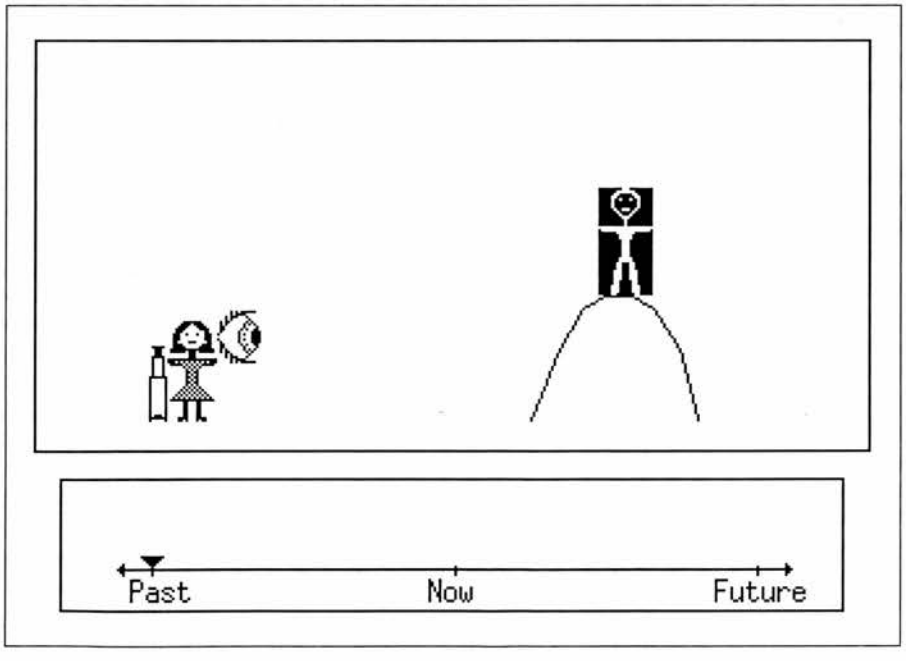
```
E,
[telescope_Device,E],
[with_Accompanying,C,E])]))]
```

These are the pictorial constraints generated for the spatial representation:

```
Spatial constraints are:

   farfrom(see_LookAt, man_MalePerson, 9)
   touching(carla, see_LookAt, 9)
   touching(man_MalePerson, hill_HighGround, 8)
   above(man_MalePerson, hill_HighGround, 8)
   left(carla, see_LookAt, 8)
   left(see_LookAt, man_MalePerson, 8)
   touching(telescope_Device, carla, 7)
   not above(telescope_Device, carla, 7)
   not below(telescope_Device, carla, 7)
   left(telescope_Device, carla, 7)
   not left(man_MalePerson, hill_HighGround, 7)
   not right(man_MalePerson, hill_HighGround, 7)
   in_top_third(see_LookAt, carla, 5)
   in_middle_horiz(man_MalePerson, hill_HighGround, 5)
```

Here is the picture for logical form 4:



Pictorial Representation for Logical Form Four

Note that the telescope is accompanying the actor, Carla. Since the constraint requires the telescope to be touching, a problem is that the telescope could be placed on the other side and interfere with the seeing event. This could cause a false implicature that the telescope was somehow being used to see the man. With the telescope on the outside, away from the seeing event, the telescope is shown to not be used. Also, there is a different icon associated with use of the telescope, but the placement for accompaniment is still very important.

5th LF for Spatial Ambiguity Example

The fifth logical form description describes the man on the hill, and Carla is *using* the telescope. Notice that the object is listed with the "_Instr" ending after the object name (i.e. telescope_Device_Instr). This is a modification of the icon to show that it is in use. One can see from the logical form that the object E, the telescope, is the instrument of the seeing event.

```
%  5th Logical Form of:
% "{Carla} saw {{the man} on {the hill}} with {a telescope}."
[dcl,
 quant(exists,
       A,
       [and,
        [man_MalePerson,A],
        quant(exists,B,[hill_HighGround,B],[on_Locational,A,B])],
       quant(exists,
             C,
             [and,
              [event,C],
              quant(exists,D,[current_time,D],[precedes_in_time,C,D])]
             [and,
              [see_LookAt,C,carla,A],
              quant(exists,E,[telescope_Device,E],[with_Instrument,C,E])
              ]))]
```

The 'with' (with_Instrument) relationship generates the following constraints:

$with(A, B) \Longrightarrow$
left(E,B),
touching(E,B),
in_middle_vert(E,B).

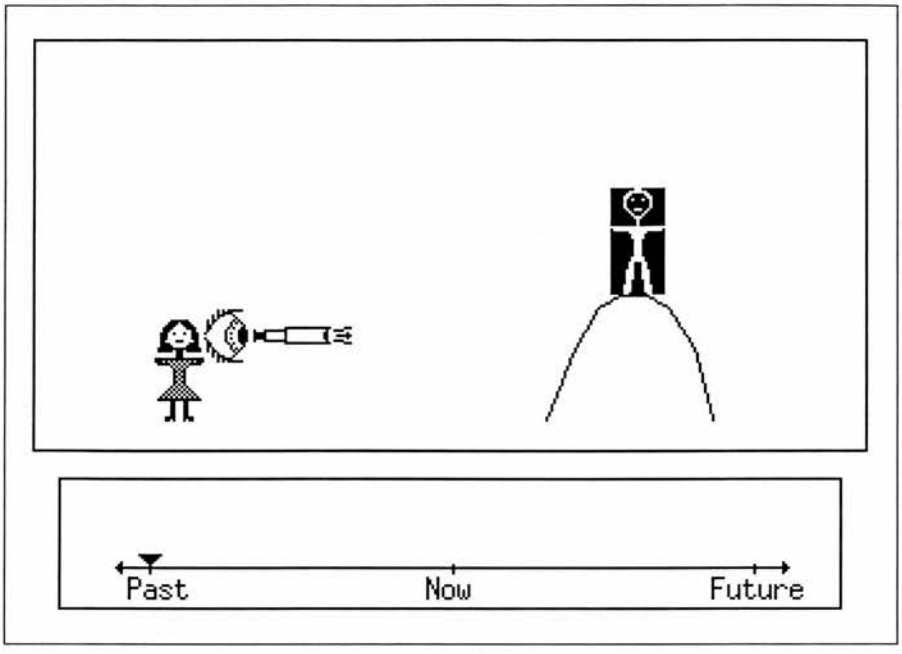
Notice that the instrument icon is attached to the event icon, rather than attached to the actor. This is consistent with the representation method of having one icon for the actor, and having a separate icon for the verb. To attach the instrument icon directly to the actor would involve domain–dependent knowledge and could not be easily generalized.

In addition to these constraints, the telescope icon is modified to become an instrument icon. This is done in the system by actually modifying the name of the icon by adding '.Instr', to indicate that it is an instrument. The following list of constraints shows the *telescope_Device* as *telescope_Device_Instr*. This is very system specific, and is only mentioned to indicate that a different icon is to be used for the instrument case.

The total set of spatial constraints is given below. Each of the different types of objects and relations that will occur in the logical form have been shown, so the constraint structure is omitted for the remaining logical forms, for ease of reading of the example.

```
Spatial Constraints are:
   touching(carla, see_LookAt, 9)
   farfrom(see_LookAt, man_MalePerson, 9)
   touching(man_MalePerson, hill_HighGround, 8)
   left(carla, see_LookAt, 8)
   touching(see_LookAt, telescope_Device_Instr, 8)
   left(see_LookAt, man_MalePerson, 8)
   above(man_MalePerson, hill_HighGround, 8)
   not left(man_MalePerson, hill_HighGround, 7)
   left(see_LookAt, telescope_Device_Instr, 7)
   not right(man_MalePerson, hill_HighGround, 7)
   in_top_third(see_LookAt, carla, 5)
   in_middle_horiz(man_MalePerson, hill_HighGround, 5)
   in_middle_vert(see_LookAt, telescope_Device_Instr, 5)
```

Here is the pictorial representation for logical form 5:



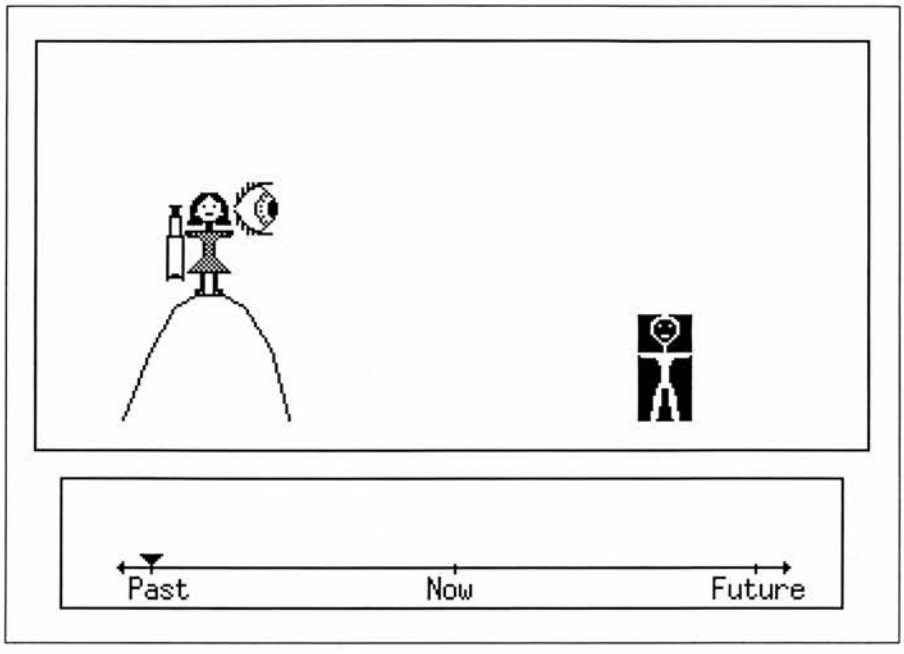
Pictorial Representation for Logical Form Five

6th LF for Spatial Ambiguity Example

The description of logical form six is that Carla is on the hill, she is holding a telescope, and sees a man without using the telescope. The logical form is:

```
%  6th Logical Form of:
% "{Carla} saw {the man} on {the hill} with {a telescope}."
[dcl,
 quant(exists,
       A,
       [man_MalePerson,A],
       quant(exists,
             B,
             [and,
              [event,B],
              quant(exists,C,[current_time,C],[precedes_in_time,B,C])]
             [and,
              [and,
               [see_LookAt,B,carla,A],
               quant(exists,D,[hill_HighGround,D],[on_Locational,B,D])],
              quant(exists,
                    E,
                    [telescope_Device,E],
                    [with_Accompanying,B,E])]))]
```

The resulting picture after the constraints are generated for logical form 6 is:



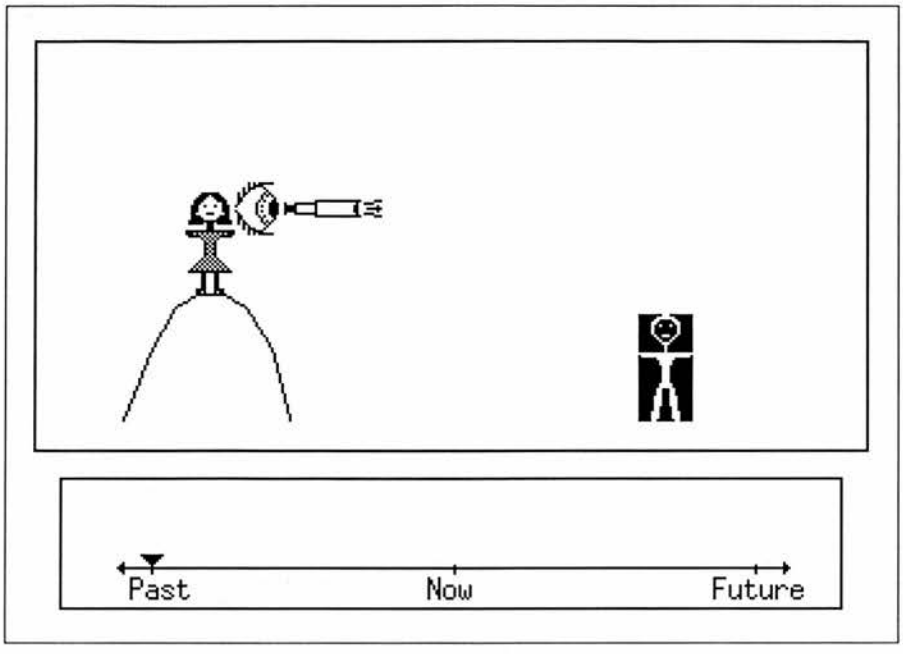
Pictorial Representation for Logical Form Six

7th LF for Spatial Ambiguity Example

The final interpretation of the sentence is that Carla is on the hill with the telescope and sees a man, but this time she used the telescope to see with.

```
%  7th Logical Form of:
% "{Carla} saw {the man} on {the hill} with {a telescope}."
[dcl,
 quant(exists,
     A,
     [man_MalePerson,A],
     quant(exists,
           B,
           [and,
            [event,B],
            quant(exists,C,[current_time,C],[precedes_in_time,B,C])],
           [and,
            [and,
             [see_LookAt,B,carla,A],
             quant(exists,D,[hill_HighGround,D],[on_Locational,B,D])],
            quant(exists,
                  E,
                  [telescope_Device,E],
                  [with_Instrument,B,E])]))]
```

Using the same technique as described before, the final pictorial representation is:



Pictorial Representation for Logical Form Seven

This concludes the spatial ambiguity example. The next section is an example of temporal ambiguity.

8.1.3 Temporal Ambiguity Example

In this example, all of the possible logical form descriptions of a temporally ambiguous sentence are shown. Sentence 81, indicates a time that is referenced, but two events are mentioned. It is ambiguous as to which event the time refers to.

(81) Alan said Ian saw the woman on Tuesday.

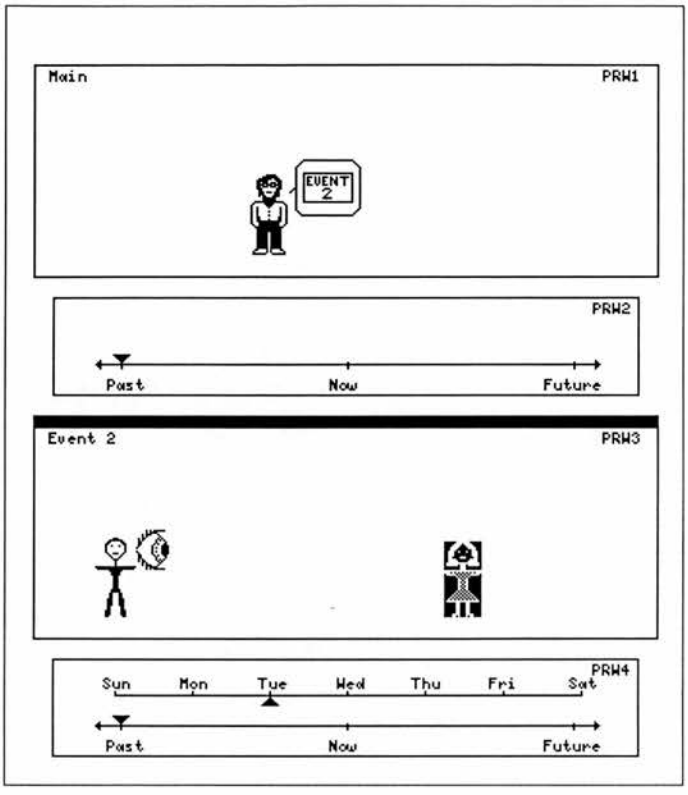
1st LF for Temporal Ambiguity Example

In the first logical form, the seeing event took place on Tuesday. The only information known about the saying event is that it took place sometime in the past. Since there are two events in this logical form, two windows are used.

The first logical form is:

```
%  1st Logical Form of:
%    "Alan said Ian saw the woman on Tuesday."
[dcl,
 quant(exists,
       A3,
       [proposition,
        A3,
        quant(exists,
              B3,
              [woman_FemalePerson,B3],
              quant(exists,
                    C3,
                    [and,
                     [event,C3],
                     quant(exists,
                           D3,
                           [current_time,D3],
                           [precedes_in_time,C3,D3])],
                    [and,
                     [see_LookAt,C3,ian,B3],
                     [on_Temporal,C3,sf(date([1992,7,7]))]]))],
       quant(exists,
             F3,
             [and,
              [event,F3],
              quant(exists,
                    G3,
                    [current_time,G3],
                    [precedes_in_time,F3,G3])],
             [say_StateThat,F3,alan,A3]))]
```

This is the first pictorial representation of the temporal ambiguity example:



PR for Logical Form One of Temporal Ambiguous Sentence

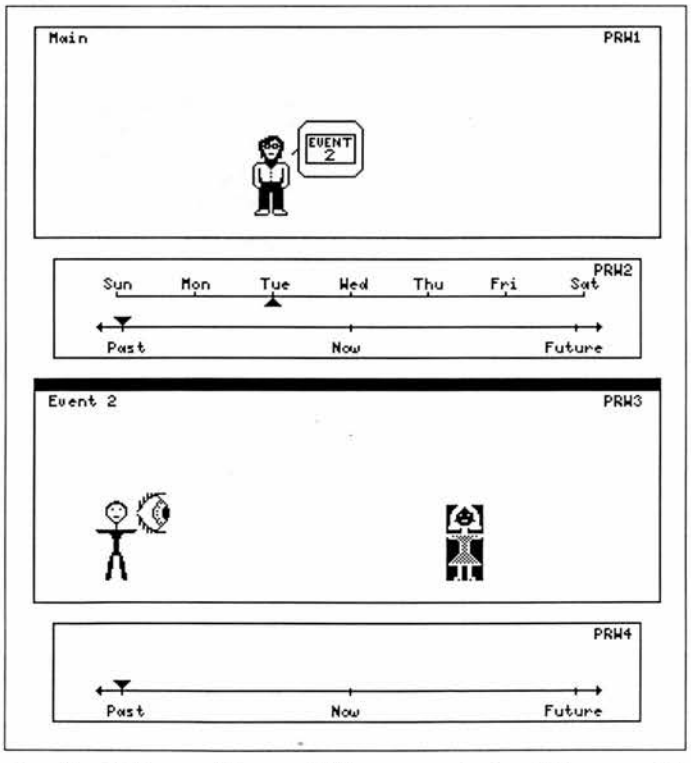
The picture shows Alan making a statement. The statement is that Ian saw the woman. Both events occurred in the past, but the seeing event specifically occurred on last Tuesday. This is showed in a temporal window, with the specific information of on Tuesday being display on the scale. The temporal information known about the event is that it simply happened in the past, and is thus displayed with the word past, without the timeline being shown in the temporal window.

2nd LF for Temporal Ambiguity Example

In the other logical form representation, the other event, the saying event, took place on Tuesday. The logical form representation contains this information which follows:

```
%  2nd Logical Form of:
%    "Alan said Ian saw the woman on Tuesday."
[dcl,
 quant(exists, A4,
       [proposition, A4,
        quant(exists, B4,
              [woman_FemalePerson,B4],
              quant(exists, C4,
                    [and,
                     [event,C4],
                     quant(exists, D4,
                           [current_time,D4],
                           [precedes_in_time,C4,D4])],
                    [see_LookAt,C4,ian,B4]))],
       quant(exists, E4,
             [and,
              [event,E4],
              quant(exists,
                    F4,
                    [current_time,F4],
                    [precedes_in_time,E4,F4])],
             [and,
              [say_StateThat,E4,alan,A4],
              [on_Temporal,E4,sf(date([1992,7,7]))]]))]
```

This is the second pictorial representation of the temporal ambiguity example:



PR for Logical Form Two of Temporal Ambiguous Sentence

This is similar to the pictorial representation of the first logical form for temporal ambiguity example. The main difference is that the statement event occurred on Tuesday, where the seeing event is only known to be in the past.

The previous two examples showed how ambiguity can be made apparent with pictorial representation. The next section looks at some of the problems with vagueness.

8.2 Vagueness

Vagueness is particularly important in pictorial representation. A picture usually requires specific information for it to be drawn. An object must be drawn in some specific location, and a specific icon must be used to represent an object. The next section discusses how vagueness and specificity depend upon the scale of representation chosen. The following section discusses the need for the missing–information module when dealing with insufficient information. Finally, the use of vagueness in the representation, by the use of pictorial generality in absence of specific textual knowledge is presented.

8.2.1 Scale of Representation

There are varying degrees of vague information, which some may not be specific enough to generate a pictorial representation. The following sentences all describe the same location, however some are more vague than others.

(82) I live at [55'58" North, 3'16" West].

(83) I live in Edinburgh.

(84) I live in Scotland.

(85) I live in Europe.

(86) I live on Earth.

Sentence 82 contains a specific longitude and latitude. Although showing locations on a map was not implemented in the system, an exact position could be drawn on a map to represent the location at the given longitude and latitude. Sentence 83, is a bit more

vague than sentence 82, but could also be represented with a point on a map if the map was of Britain for example. This example shows that the degree of specificity is dependent upon scale of map on which to represent the information.

The information in sentence 84 may be considered very vague if it involved a conversation involving two people who live in Edinburgh. However, the same information may be seem quite specific, when someone asks "do you live in Britain?", and the speaker replies "yes, I live in Scotland." Since this system involves pictorially representing the text, a scale must be chosen for the pictorial representation. The scale that is chosen is one that makes the statement specific. This assumes that the text is giving specific information. This assumption will offer pictures that are more interesting, rather than show pictorial representations that are correct, but do not imply much information, nor the intended information.

The scale of the map that makes a statement specific, is the one that shows the location as a "point." This will rule out all of the scales that are very detailed and thus would require more information for exact placement on the map. Among the set of scales which will all show a certain location as a point, the scale that is the largest (which shows the most detail) is chosen. Therefore, sentence 83 could be shown on a map of Britain, which shows Edinburgh as a point. It could not be shown as a point on map of Lothian region[3], or on a map of Edinburgh city, because the location could not be shown as a point on these maps. So those scales are not selected. However, Edinburgh could be listed as a point on a map of Europe, or on a map of the world. The map of Britain is chosen because it is of larger scale than the maps of Europe or the world,and therefore, is a representation that shows the most specific information.

When generating a pictorial representation, a spatial scale may or may not be required. If it is not required, it should not be displayed. However, if it is required, then the technique of determining a scale is to find the scale that both:

- shows the textual information as *specific*, (and at the same time);
- selects the biggest scale (one containing the most detail).

[3] This is the region, county, or shire that contains Edinburgh.

8.2.2 Insufficient Information

In section 4.3.4, the Missing-Information Module was introduced. This module supplies information when the information gleaned from the logical form is insufficient. In the example, of a cup on the table, the specific location of the cup on the table is not known. Obviously, the pictorial representation needs a specific location. This module adds location information in absence of specificity by supplying additional constraints of a low priority. If specific information was previously supplied from the logical form, that specific information will generate a constraint of a high priority, and not be affected by the lower priority constraint.

The rules in the Missing–Information Module are simple, but are required. This module can be modified by the user, and he or she may add more domain-dependent rules. The rules include:

- Do not allow objects to touch, if not specified.
- Do not allow objects to be near each other, if not specified.
- If objects are touching, center them, in the absence of other locational information.
- If a spatial scale must be chosen, choose the scale to draw the location as a point.
- If multiple scales are mentioned, then use multiple pictorial representation windows, each with their own scale.

If the Missing-Information Module makes an incorrect assumption, the resulting pictorial representation could lead to a false implicature. Therefore, these rules attempt to reduce the chance of supplying too much specificity, especially the case of supplying incorrect information.

8.2.3 Vague Representation

The final technique is to try to use vague pictorial representations when the textual information is vague. In the previous temporally ambiguous sentence, it was known that one of the events took place on Tuesday, while the other event occurred sometime in the

past. The temporal window for the event that occurred on Tuesday showed a scale of the days of the week with the pointer pointing to Tuesday. This implies specific information. In the other temporal window, is the word "past." Originally, when the system was first developed, it had a scale which showed a "past-present-future" scale, with the pointer pointing to past. However, if two events were *pointing* to the past marker on a *scale* this seemed to indicate that the two events occurred at the *exact* same time in the past. What was meant to be indicated was the two events did not take place in the present or the future. No information relating the times of the two events was intended. Therefore, the scale for "past-present-future" is not used, unless it is known that although both events were in the past, and there is some ordering among those events. Notice that this is the case in the pictorial representations for the two logical forms of the temporal ambiguous sentence.

This chapter described the implemented application of the pictorial representation concept to disambiguate sentences. The previous chapters presented a pictorial representation of several varied natural language expressions. This completes the discussion of the implemented system. The next chapter describes two possible applications of the pictorial representation concept.

Chapter 9

Possible Applications

This chapter proposes two possible applications of the pictorial representation concept. The first application is a data fusion technique to pictorially represent data from a *large* number of natural language texts. Several text processing systems are now attempting to handle text from magazines, newspaper articles, et cetera. This may be a method for a user to make sense of the overwhelming amount of data contained within these textual sources. The second application is to use pictorial representation as an inter–lingua. A simple proposal of representing airline flight information from several languages pictorially is given. It is not meant to be a machine translation technique, but could be a useful application in several domains.

9.1 Data Fusion Pictorial Representation

The thesis describes the pictorial representation concept for a single sentence, but how can the concept be extended to work for discourse within a single textual message? And how can the concept be extended to represent data from several textual messages?

The main technique involves using an icon representation of complex data. I propose a user–defined association of graphical icons with the text stored in Meaning Representation Slots [Ludlow 88]. The interactive graphics display subsystem allows the user to list the "variables" to be displayed and to list a time of interest or area of interest (spatially). The association or binding between the icon and text in the meaning representation will be accomplished by a menu. This is where the domain-dependent knowledge is defined. This association or binding is not simply a binding of icons to an object, rather a sophisticated

combination of any of the meaning representation slots.

Figure 9.1 is a sample text of multiple events.

```
   Several English football hooligans attacked Italian spectators
at a football match between Manchester United and the Italian
national team held on Saturday (June 20). Police arrested three
men who were involved in a stabbing.
```

Figure 9.1: Sample Text of Multiple Events

Figure 9.2 is a possible menu display. The variables are associated with aliases and a graphical symbol or icon. These associations will be made by using this menu display. The example association is one for the variable "attacks," and the data base containing British police messages. In Figure 9.2, the variable is associated with four different icons and the associated "meaning." A knife icon is used if the main verb slot contains (stab+) or the instrument slot contains (knife).

Some domain-dependent knowledge is required for the text-to-pictures process, particularly in this icon to variable assignment definition. What is noteworthy, is that the text processing portion heavily relies upon domain-independent knowledge. Where part of the pictorial process, on the other hand, relies upon domain-dependent knowledge. This split of knowledge allows a single text processor to format a large set of textual data into one large event data base. Many different users (analysts) can use the same large event data base to analyze their particular topic. Each user will define the subject domain-dependent knowledge needed to display their subject. The pictorial processor displays the data in the large event data base via the "interpretation" provided from the user defined domain-dependent module. The actual graphics display is determined by their choice of variables to be plotted and what area of interest.

The variable icon binding is a form of data abstraction. It performs a data abstraction by allowing a boolean combination of several linguistic characteristics to be represented by a single icon. This is a form of pictorial semantics on top of logical form representation. The next section shows a simple example of how some fusion of data could be displayed.

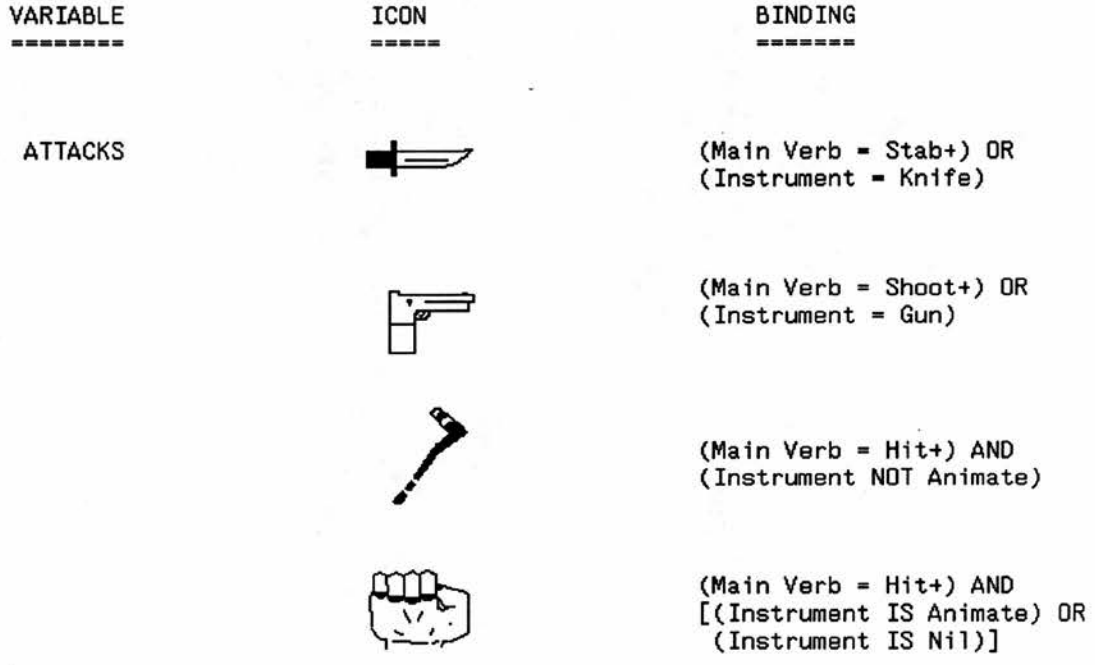


Figure 9.2: Possible Menu Display of Variable Icon Binding

9.1.1 A Sample Data Fusion Graphical Representation

After the variables, type of display, and area of interest are chosen, the proposed system would generate the appropriate query to the data base (stored logical forms representing the messages). Only the events that correspond to the query are displayed.

Figure 9.3 shows a possible display of police messages. Three variables (informants, attacks, and arrests) are plotted versus a time-line. The knife icon is shown under the attack variable to represent an event of an attack where a knife was used. The knife icon is also placed correctly along the time–line. The event described in Figure 9.1 occurred on June 20 and is shown on the display. Notice also the arrests are shown on June 20. Many police messages could be stored in the data base. Some of these other messages are shown—by the events listed by the gun icon and the fist icon.

Notice the time–line is only for the month of June. When the display type is chosen (which variables are plotted against what) and a time of interest is chosen (in this case only June), the system queries the event data base to find if any of the textual messages contain those variables and are in the appropriate time of interest. Only the events that fit these criteria are plotted. Therefore, a *filtering* process occurs.

I believe filtering is required. The event data base could contain hundreds of messages. Each message can contain several events. If all the events were displayed, the display may not be of value to the analyst. At least, the analyst has the option to specify how wide or narrow they want the area of interest. This ability to allow the user to set the filter *bandwidth* should help in data integration.

There are several possibilities of display with this proposed system. Expansion of the system and areas of experimentation are listed next.

9.1.2 Further Extensions of Data Fusion Application

The key area here is to investigate techniques of displaying large amounts of data with sophisticated graphic symbols that quickly allow the user to discern the information. Certain displays may allow multiple presentation of data. Map displays with time–line displays could be combined.

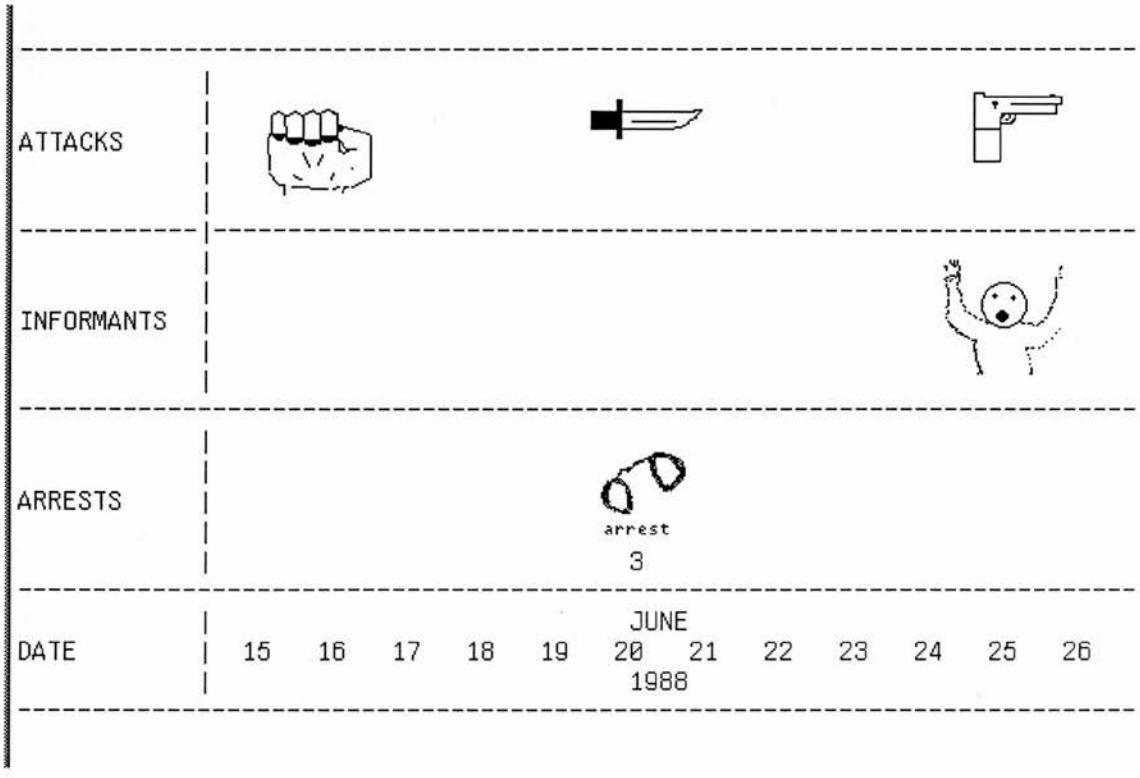


Figure 9.3: Possible Graphical Representation of Police Messages

Another area of research is how to correlate events that are related by time, location, or some common element. Some type of situation marking or modeling may be required. Cybernetics systems, or Situation Modeling [Pospelov 86] may provide a framework for "viewing" the data.

Multi–representational techniques should also be exploited. A proposed display would be one where the data could be represented both temporally and spatially. The information can be plotted on a map with a time-line scale along the bottom. The marker on the time-line scale can slide back or forward through time with movement of the mouse. Event representations will turn on or off from the display if they occurred during the appropriate time. The width of time marker can also be adjusted by the mouse. Maybe the width was only "one week." The user could adjust to "one year" to see all of the events in the last year. Charnoff faces are another technique of representing multiple amounts of data. Charnoff faces use the display of several human faces with different facial characteristics that represent different data [Pospelov 86]. Although Charnoff faces were only used for statistical data, similar techniques could be employed in this system.

9.2 System Use with Other Languages

A picture is the same in any language. A pictorial representation of text can cross the language barrier. This section describes an application which is an alternative to a machine translation system by making use of the "Text to Pictures" concept.

This application is not intended to be a panacea for the general machine translation problem, rather this method is put forward as an alternative for handling the specific problem of understanding narrative text of several source languages. The application converts the text directly to pictures, rather than "translating" the text of a source language into text of a target language.

This section proposes how the text–to–pictures technique can be extended to be used for a machine translation problem by converting text of several source languages to a single picture. An example using six source languages is given in the next section.

(87) A large hurricane destroyed several buildings in London.

The language processor may recognize, in the example sentence 87, that "in London" is a location. For this event, "in London" is stored in the "location" slot. But there is no knowledge of actually where London is. The pictorial generation system would contain this domain-dependent knowledge. The graphics system then can place a icon representing a "hurricane" over the location on the map where London is located.

The association of the icons with the meaning representation slots was described in the last section about data fusion.

Using the concepts presented throughout the thesis, showing how to pictorially represent one language, an extension for several input languages will be shown. Figure 9.4 shows the general system. Each narrative source language has a language processor that identifies the pieces of the input sentence that correspond to meaning representation slot. For an English sentence that contains the fragment "in Moscow," the fragment "in Moscow" would be stored in the location slot. If it was a Russian sentence that contained "*b Mockba*," again this fragment "*b Mockba*" is stored in the location slot. The Meaning Representation Slots contain the original source text–the text is not converted to another language.

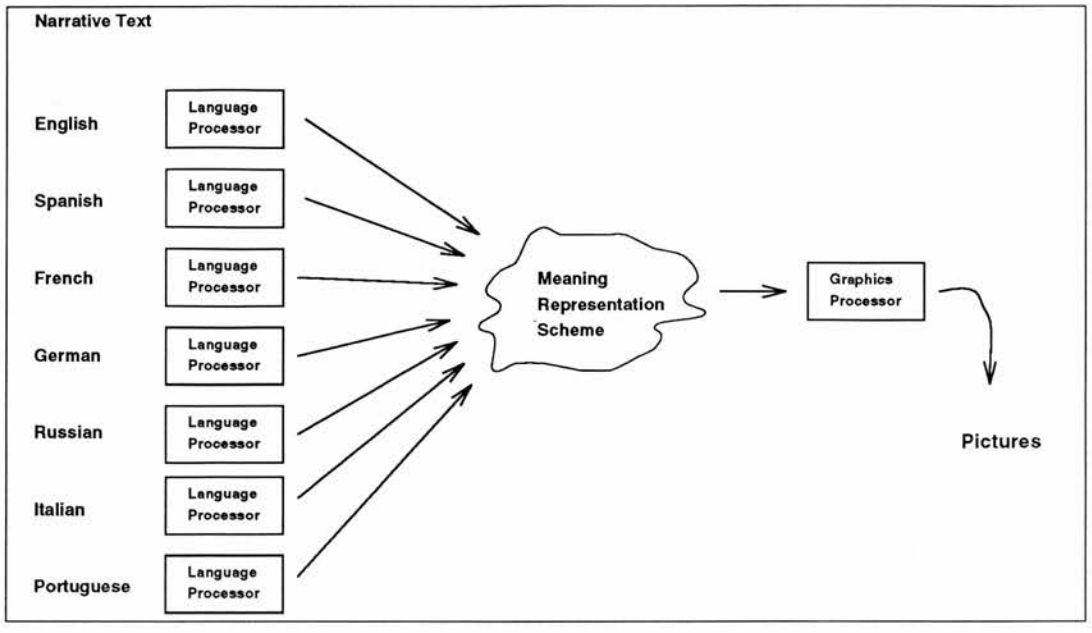


Figure 9.4: Translating Several Languages into One Picture

Figure 9.4 has seven language processors that "break up" the source text into the corresponding meaning representation slots. Although the text is categorized into different meaning slots, it still is in seven different languages. So how does one get a single picture from this representation?

Figure 9.4 only shows one graphics processor. The graphics processor uses a user-defined mapping between the icon and the text stored in the slots. Figure 9.5 shows the variable-icon binding that is needed for an example of an icon for an "airplane."

9.2.1 Example with Seven Languages

An example of using this mapping of aircraft is shown in figure 9.6. Seven source texts are listed describing European flights to Berlin. A translation of each source text to English is provided in this paper, but is **not** used in the actual text to pictures system.

The general text to pictures approach and how it could be extended to accept several source languages was shown. Many other pictorial representations are possible besides a map. Time-line displays can show when certain events took place. Graphs or charts can depict the number of objects. Color can be used to represent the color of an object or to

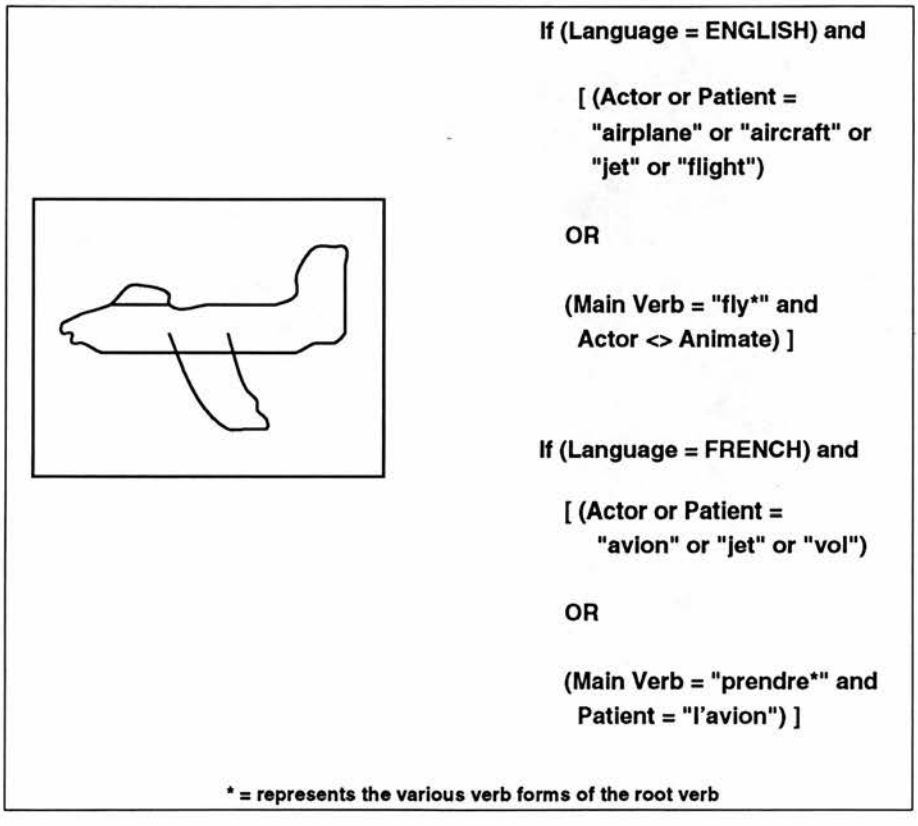


Figure 9.5: Example of User Defined Multi-Lingual Mapping to One Icon

English Text

British Airways has two direct flights to Berlin.

Spanish Text

Iberia no tiene vuelos directos a Berlim.
Iberia does not have direct flights to Berlin.

French Text

Air France a augmente le nombre de ses vols sur Berlin. Ils proposent maintenant 4 vols par jour au depart de Paris.
Air France increased its service to Berlin. They now have four daily flights from Paris.

German Text

480 passagiere flogen von Munchen nach Berlin mit den drei linienflugen.
480 passengers flew from Munich to Berlin on the three scheduled flights.

Russian Text

Eta dva covalot b Mockba c Berlin.
There are two flights from Moscow to Berlin.

Italian Text

Alitalia offre un volo giornaliero per Berlino. Transportano in media 150 passeggeri per volo.
Alitalia has one daily flight to Berlin. They carry an average of 150 passengers per flight.

Portuguese Text

TAP tem dois voos por dia para Berlim.
TAP has two flights per day to Berlin.

Figure 9.6: Example of Seven Languages

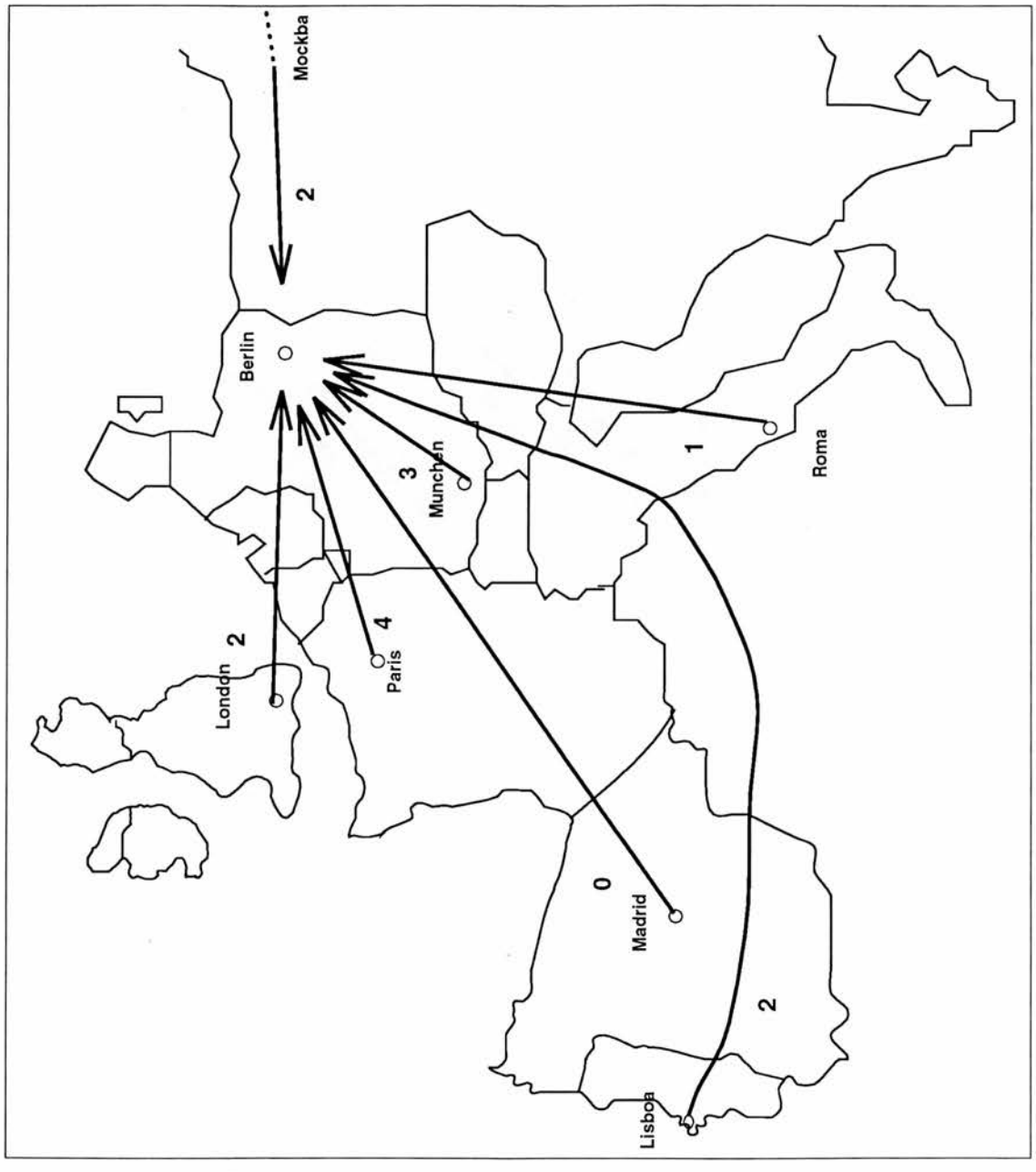


Figure 9.7: Map Showing Flight Information

represent the intensity of an event. The graphical representations are numerous, but all of them convey meaning across the language barrier.

Again this approach cannot solve the general machine translation problem, but may provide a technique for problems that require "analyzing" large amounts of narrative text from several languages. Such an approach may also be able to depict simple sentences as a teaching aid to display pictorial meanings of nouns, verbs, and prepositions to foreign language students or children.

Chapter 10

Conclusions

10.1 Summary

This thesis presents a concept, a working system, and examples of pictorial representation that can represent several varied natural language expressions.

A background of the relevant text–pictures systems is given. The major limitation of the existing systems is that they primarily only handle objects and some spatial expressions. A text processing system is critical to an overall working Text-To-Pictures System, and several are identified.

The Pictorial Representation Concept is described. A terminology is defined. The fundamental idea is that text can be represented by a visual scene which contains icons that represent objects, and Pictorial Representation Windows (PRWs) that can display the icons and a set of relations. The set of relations are displayed by location of objects or other PRWs, inclusion of new objects, or modification of existing objects.

A working Text-To-Pictures System's design is given. It makes use of the Core Language Engine as the text processor. The logical form descriptions of the text are translated into a Constraint Satisfaction Problem. This is accomplished by representing objects with icons, and representing relations via a set of pictorial primitives. The entire process of the system is described in detail.

Several general expressions are pictorially represented. The object concept is described which allows an icon to represent an object. Size of objects are represented by a logarithmic technique of scaling icons relative to others. The size of icons is captured within

the data structure of the CSP. Number is implemented in the system via two different techniques of representing the actual number of objects or superimposing a number over the object to indicate a large number of objects. Relative Clauses and Embedded Sentences are both pictorially represented by showing one event in one pictorial representation window, and the other event in another adjoining pictorial representation window. Accompaniment and Instrument cases are both represented by using placement and modification of objects.

The use of the Pictorial Representation Window allows conjunction and disjunction to be represented. Negation is included in the system, and negates the correct scope of the text by reverse video of the appropriate pictorial representation window. Distributive and Collective Scoping is handled in the system as well.

The Subject-Verb-Object structure is used as a default of placing the subject left of the verb, and the object to the right of the verb, in the absence of other locative information.

Spatial and Temporal Expressions are described in detail. A set of seventeen spatial primitives and five temporal primitives are given. Spatial Primitives to handle above, below, left, right, touching, near, far, between, overlapping, and "in-third-of" regions are defined explicitly. Temporal Primitives of begin, end, before, after, and duration are defined. Negation can be applied to each of these primitives. Examples of each of the primitives being applied are given for various expressions. Selection of viewpoint is discussed. Special modules such as Naive Physics and Missing Information Modules are described.

Using a pictorial definition simplifies the spatial expression representation problem. Using spatial primitives combined with naive physics gives a powerful mechanism for representing spatial expressions. This technique is an improvement over the work of Vandeloise, the Toulouse Group, and Herskovits, in that this framework of representing spatial expressions as a constraint satisfaction problem allows a working system to be developed.

The system can display the many possible interpretations of an ambiguous sentence. This can allow a user to choose the intended meaning of a sentence. The system is demonstrated with a spatially ambiguous sentence and a temporally ambiguous sentence. The problems with vagueness are addressed.

Finally, two possible applications of the pictorial representation concept are given. One is a data fusion technique to pictorially represent large amounts of textual data succinctly. The other application is using pictorial representation as an inter-lingua.

10.2 Re-address Thesis Issues

There are several questions that are important to the issue of the inter–relationship of text and pictures. The main questions that were addressed in thesis report were:

1) **Can a pictorial representation be developed to adequately represent text?**

Several varied natural language expressions were pictorially represented. In addition to objects, and some simple spatial expressions, object size and modification of objects were represented. Number, Relative Clauses, Embedded Sentences, Accompaniment Case, Instrument Case, Conjunction, Disjunction, Collective Scoping, Distributive Scoping, SVO structure, Negation, and several varied spatial and temporal expressions were all represented.

Over twenty subjects used the working system to see if they could interpret the intended textual sentence from the generated visual scene. Their feedback and comments were implemented throughout the development of the working system, which was described in section 4.4.

2) **Can the inter–relationship between natural language text and pictorial representation be described?**

Chapter three of this report describes my pictorial representation concept and gives a formal description. A set of pictorial primitives is given that formally defines the inter-relationship between natural language text and pictorial representation. By using a Constraint Satisfaction Problem approach, the system converts several natural language expressions into a pictorial representation.

3) **If the inter–relationship can be described formally, can a working system be built using this description?**

A *working system* was shown to use the translation process from text to a pictorial representation. The system design was fully described in chapter 4 of the thesis. Examples

of the working system were shown with varying linguistic expressions.

4) **In detail, *how* are specific types of natural language expressions converted to a pictorial representation?**

The translation process for each different (linguistic) type of expression to a pictorial representation is different. Chapters 5 through 8 addressed the conversion processes in detail. Again, many varied natural language expressions were represented–much more than by any previous method.

5) **What knowledge (domain dependent or domain independent) is required for the conversion process of text to pictures?**

When designing a system, transportability is an issue. Some domain–dependent knowledge is required for the conversion process. The system design attempts to isolate domain–dependent knowledge modules from domain–independent knowledge modules. The domain–dependent knowledge that is needed in the system, are the modules shown in figure 4.1, that the user has access to. These modules include the pictorial grammar (but not the pictorial primitives), the Naive–Physics Module, Missing–Information Module, and the Icon Library.

10.3 Major Contributions of Thesis

The first contribution of this thesis is that it constitutes the first report that combines the work on the inter–relationships between text and pictures. There are several contributions from early text/pictures systems, text processing, representation schemes, natural language processing, graphics, mathematics, and artificial intelligence. It is the combination of all of these topics that allows the topic of the inter–relationships between text and pictures to be addressed.

The second contribution is the extension of pictorial representation to handle a wide range of natural language expressions. Previous work only included objects, and some spatial expressions. This work included noun modifiers, temporal expressions, conjunction, quantification, and some verb features.

Another contribution is the discussion of the relevant issues concerning converting text to

pictures. A terminology is defined. Problems of vagueness and ambiguity are addressed. Topics such as data fusion and using pictorial representation to represent several languages are discussed.

A notable contribution was the building of a working system. Although a small system, it demonstrates what the required components are in the conversion process from text to pictures. This system allowed the thesis topics to be addressed via experimentation. The system also allows other researchers to expand their research areas without having to worry about other components that are required for the overall text–to–pictures process.

Finally, the design aspects of the implemented system are a contribution to text-picture systems. The system is based on translating the logical form description of the text into a Constraint Satisfaction Problem (CSP). A set of pictorial primitives is defined, and several varied components of natural language expressions are described using the set of primitives in a principled way. Real pictures are generated by the Text-To-Pictures System and are included in the thesis to illustrate the system's capability.

10.4 Areas for Further Research

The approach of representing one sentence may be able to be expanded to handle more than one sentence. Pictorial representation of discourse involves many of the same problems that text processing involves. The pictorial representation could be extended even further via data fusion techniques as described in previous chapter. This approach would have to make use of text summarization techniques.

The text–to–pictures system itself can be improved. It was developed to be an academic tool to defend the ideas presented in this thesis. Since the system was designed to be modular, modules could be replaced with more powerful or useful modules. To make the system a more useful product, the pictorial generation process software and the language engine should run on the same computer. The pictorial generation system could be added to the Core Language Engine project as a tool for pictorially representing ambiguous sentences to non–linguist users. Also a commercial three-dimensional graphics package should be implemented to make use of the *in-front-of* and *behind* constraints and to present a more interesting picture.

Finally, the system can be used to explore definitions of spatial and temporal expressions to generate a large list of useful definitions of these relations. A picture quickly tells the researcher if the definition has omitted some constraint or if the constraint is too restrictive. Hopefully, the area of natural language systems that interact with graphics will continue to grow.

Appendix A

Main Text–To–Pictures Routines

```
%%=============================================
%% loader
%%
%%      Loads the files for the Text-to-Picture System (TTPS) Program
%%
%%
%%---------------------------------------------
%% This software may be used for educational and research purposes.
%% It is distributed in the hope that it will be useful, but
%% WITHOUT ANY WARRANTY. No author or distributor accepts any
%% responsibility to anyone for the consequences of using this code.
%%
%% You may freely distribute/modify this software provided this whole
%% comment remains intact.  You may not sell this software.
%%
%% (c) 1992 Nelson D. Ludlow, University of Edinburgh
%%=============================================

%-----------------------------------------------
% Load standard Prolog Routines
%-----------------------------------------------

:- ['prolog.ini'].                  % standard prolog init routines
:- ['prologa.ini'].                 % arity compatible prolog routines
:- ['prologs.ini'].                 % Sepia Prolog retract fix

%-----------------------------------------------
% Constraint Builder Modules
%-----------------------------------------------

:- ['cbuilder.cbm'].                % main
```

```
% Constraint Builder Grammar for the Sentence
%
                                    %(Constraint Defns)
:- ['sprimitv.cbm'].                %spatial primitives
:- ['tprimitv.cbm'].                %temporal primitives

:- ['sentence.cbm'].                % build constraints from LF
:- ['parself.cbm'].                 % parses the lf

:- ['verb.cbm'].                    % build verb constraints
:- ['spatial.cbm'].                 % build spatial expressions constraints
:- ['temporal.cbm'].                % build temporal expressions constraints


% Other Constraints Modules
%
:- ['nphysics.cbm'].                % naive physics constraints
:- ['nicepict.cbm'].                % nice pictures constraints
:- ['missinfo.cbm'].                % missing information constraints


% Other Modules and Routines
%
:- ['userdef.cbm'].                 % user defined variables
:- ['visobj.cbm'].                  % visual object routines
:- ['normalzc.cbm'].                % normalize constraints
:- ['dstructs.cbm'].                % data structure routines


%-----------------------------------------------
% Load the CSP Solver files
%-----------------------------------------------

:- ['cspsolve.csp'].                % CSP Solver using Chip
:- ['captsols.csp'].                % Capture Solutions fm Chip


%-----------------------------------------------
% Load the Icon Library
%-----------------------------------------------

% Note, the graphics are handled in a seperate program.

:- ['icon.lib'].                    % The Icon Library

%-----------------------------------------------
% Top Level of TTPS
%-----------------------------------------------
```

```
:- ['toplevel.sys'].                    % Top level of TTPS
:- ['initialz.sys'].                    % Initialize the Program

%-----------------------------------------------
% Read in the Logical Forms to process
% (INPUT DATA)
%-----------------------------------------------

:- ['example.lfs'].                     % Example LF Data Base
```

```
%%=============================================
%%  toplevel.sys
%%
%%      Top Level of TTPS
%%
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================

ttps(INPUT,OUTPUT) :-
            %initialize stuff(for crappy SEPIA/CHIP)
            initialize,

            %Read in the LFs to be processed
            nl,nl,
            write('Searching for LFs...'),nl,
            slfs(INPUT,Sentence,NumLFs,SLFS),
            write('For the sentence:'),nl,
            write(Sentence),nl,nl,
            write('There are '),write(NumLFs),
            write(' LFs for this sentence.'), nl,

            %generate a Pictorial Repr for each LF
            loop_solve each_lf(NumLFs,SLFS),

            nl,nl,
            write('All LFs have been processed for this sentence.'),

            % organize the solutions into one file for graphics
            % on another system
            capture_solutions(OUTPUT,Sentence),

            %uninitialize stuff(for crappy SEPIA/CHIP)
            uninitialize.

%-----------------------------------------------
%  Loop to Solve and Display Each LF
%
%       There are three possiblities
%          a) all Lfs have been handled
%          b) normal case
%          c) for some reason couldn't draw the LF
%             so gracefully go on to next LF
%
%
%-----------------------------------------------
loop solve_each_lf(NumLFs,SLFS) :-
```

```
    handleLF(1,NumLFs,SLFS), !.

%
% termination of loop
%
handleLF(NumCurrLF,NumLastLF, EmptyLFlist) :-
    NumCurrLF > NumLastLF, !.

%
% Normal Case
%
handleLF(NumCurrLF,NumLastLF,[CurrLF|RemLFs]) :-
      NumCurrLF =< NumLastLF,
      nl,nl,
      nl,write(' ------------------------------'),
      nl,write(' Begin Processing LF Number '),write(NumCurrLF),
      nl,write(' ------------------------------'), nl,

      %-----
           % Handle each PRW, could be multiple windows
           % initialize to window 1
           NumPRW = 1,
           retractall(subwindow(_,_)),
           handlePRW(NumCurrLF, CurrLF, NumPRW,"'Main'"),

      %-----
      NewCurrLF is NumCurrLF + 1, !,
      handleLF(NewCurrLF,NumLastLF,RemLFs).

%
% if one LF failed keep going on to next
%
handleLF(NumCurrLF,NumLastLF,[_CurrLF|RemLFs]) :-
       NewCurrLF is NumCurrLF + 1,
       handleLF(NewCurrLF,NumLastLF,RemLFs).



%-----------------------------------------------
%     Handle Each Pictorial Representation Window
%
%          could be multiple windows for subclauses
%-----------------------------------------------
handlePRW(NumCurrLF,CurrLF,NumPRW,WinTitle) :-

           % BUILD LF CONSTRAINTS
           build_lf_constraints(CurrLF,SpatCS,TimeCS,POL),!,
```

```
% START UP THE CSP SOLVER
nl, write('Starting up the CSP Solver...'),
csp_solve(POL,SpatCS), !,

% NAIVE PHYSICS CONSTRAINTS
nl, write('Applying Naive Physics...'),
naive_physics_constraints(POL), !,

% NICE PICTURES
nl, write('Applying Nice Pictures...'),
nice_pict_constraints(POL), !,

% MISSING INFORMATION
nl, write('Applying Missing Information...'),
missing_information_constraints(POL), !,

% OUTPUT/SAVE SOLUTION
nl,nl,write('SOLUTION is:'),nl,
(retract(pr_solution(NumCurrLF,OtherWindowSols)) ->
     OtherSols = OtherWindowSols; OtherSols = []),
reverse(POL,RPOL),
append(OtherSols,[NumPRW,WinTitle,TimeCS,RPOL],FullSols),
assert(pr_solution(NumCurrLF,FullSols)),
s_write(POL),s_write(TimeCS),nl,

% CHECK FOR OTHER PRWs NEEDED
(retract(subwindow(NextWinTitle,NextWinLF)) ->
     NextWinNum is NumPRW + 1,
     nl,write('Handling SubWindow '),write(NextWinNum),write('...'),
     handlePRW(NumCurrLF,NextWinLF,NextWinNum,NextWinTitle)
; true).
```

Appendix B

Constraint Builder

```
%%=============================================
%% cbuilder.cbm
%%
%%      Main file for Constraint Builder
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================

%---------------------------------------------
% variable descriptions
%      LF       - logical form description for sentence
%      LF_Data  - Useful data from LF
%      S_TCS    - Spatial Total Constraint Structure
%      T_TCS    - Temporal Total Constraint Structure
%      S_POL    - Spatial Pictorial Objects List
%---------------------------------------------
build_lf_constraints(LF, S_TCS, T_TCS, S_POL) :-
    retractall(lfi(_)), !,                    % Clear Old LF info from DB

    nl, write('Parsing LF...'),
    parse_lf(LF), !,                          % Put New LF info to DB

    nl, write('Building Constraints from LF...'),
    cbuild_lf(CS1,S_POL), !,                  % Build Constraints from LF
                                              % and Build Pictorial Obj List
    nl, write('Normalizing Constraints...'),nl,
    normalize_constraints(CS1,S_TCS,T_TCS), !.   % Ordered by priority
```

```
%%=============================================
%% sentence.cbm
%%
%%      Builds a constraint stucture for a sentence
%%      given an LF
%%
%%      The constraints are build via
%%      a semantic grammar to describe
%%      a declarative sentence using a
%%      set of graphical constraint primitives.
%%
%%      The pictorial constraint primitives
%%      are in a two files sprimitv.cbm
%%      and tprimitv.cbm
%%
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================

%---------------------------------------------
% Sentence defintion of constraints
%      LF_CS - constraint structure from the LF Data
%      VO    - constraint structure for Visual Objects
%      NVO   - new and final CS for Visual Objects
%      LFD   - Logical Form Data
%      Objs  - List of Objects in LF
%      Rels  - List of Relationships in LF
%---------------------------------------------

cbuild_lf(LF_CS,NVO) :-
   get_lf_data_from_db(LFD),  !,          % get LF parsed data
                                          % from Data Base

   nl, write('LF data from db is '),write(LFD),nl,nl,

   find_obj_and_rel(LFD,Objs,Rels),!,     % split LF Data list into to 2 lists
                                          % List of Objects and List of Relations

   nl, write('Objects are '),write(Objs),
   nl, write('Relations are '), write(Rels),

   make_visual_objects(Objs,VO), !,       % Build Data Structure for
                                          % the Visual Objects

   nl, write('Visual Objects are '), write(VO),

   cbuild_relats(Rels,VO,NVO,LF_CS), !,   % Build Constraints from the
```

```
                                        % List of Relationships

    nl, write('*****************************************'),
    nl, write('Total CS is '), nl,s_write(LF_CS),nl,
    write('*****************************************'),nl.

%-----------------------------------------------
%  Get LF Data from Data Base
%-----------------------------------------------
get_lf_data_from_db(LFD_List) :-
        get_lf_data([],LFD_List).
get_lf_data(OList,NList) :-
        lfi(LF_Data),
        retract(lfi(LF_Data)),
        get_lf_data([LF_Data|OList],NList).
get_lf_data(L,L).

%-----------------------------------------------
%  Split LF Data List into Objects and Relationships
%-----------------------------------------------
find_obj_and_rel(LFD,Objs,Rels) :-
        categ_lfd(LFD,[],Objs,[],Rels).
categ_lfd([],O,O,R,R).
categ_lfd([D|Rem],OObjs,TObjs,ORels,TRels) :-
        is_lf_object(D), !,
        categ_lfd(Rem,[D|OObjs],TObjs,ORels,TRels).
categ_lfd([D|Rem],OObjs,TObjs,ORels,TRels) :-
        categ_lfd(Rem,OObjs,TObjs,[D|ORels],TRels).

%-----------------------------------------------
%  Is it an OBJECT ?
%-----------------------------------------------
is_lf_object([OName,OVar]) :-
        atom(OName),
        atom(OVar),
        string_search(1,'obj',OVar,0).

%-----------------------------------------------
%  Make the Visual Object Records
%-----------------------------------------------
make_visual_objects(Objs,VO_CS) :-
     make_visual_objects(Objs,[],VO_CS).
make_visual_objects([],CS,CS).
make_visual_objects([Obj|RemObjs],OCS,NCS) :-
     mvo(Obj,OCS,ICS),
     make_visual_objects(RemObjs,ICS,NCS).

mvo([Descr,Tag],Old_VO_CS,New_VO_CS) :-
```

```
    make_visual_object(Descr,OCS),
    New_VO_CS = [pictorial_object([Tag|OCS])|Old_VO_CS].

%-----------------------------------------------------------
%    Build the Constraints
%-----------------------------------------------------------
cbuild_relats(Rels,VO,NVO,LF_CS) :-
     cbuild_verb_relations(Rels,VO,VO1,[],LF_CS1), !,
     cbuild_other_relations(Rels,VO1,NVO,LF_CS1,LF_CS).

cbuild_verb_relations([],VO,VO,CS,CS) :- !.
cbuild_verb_relations([Rel|RemRels],VO,NVO, OldCS,NewCS) :-
     cbg_verb_relationship(Rel,VO,IVO, OldCS,IntCS), !,
     cbuild_verb_relations(RemRels,IVO,NVO, IntCS,NewCS), !.

cbuild_other_relations([],VO,VO,CS,CS) :- !.
cbuild_other_relations([Rel|RemRels],VO,NVO, OldCS,NewCS) :-
     cbg_other_relationship(Rel,VO,IVO, OldCS,IntCS), !,
     cbuild_other_relations(RemRels,IVO,NVO, IntCS,NewCS), !.

cbg_verb_relationship(Rel,VO,IVO, OldCS,IntCS) :-
     verb_defs(Rel,VO,IVO, OldCS,IntCS,Actor),
     Rel = [_Verb,Event,Actor_Data,Patient], !,

     retractall(actor(_)),
     retractall(event(_)),
     retractall(patient(_)),

     assert(actor(Actor)),
     assert(event(Event)),
     assert(patient(Patient)), !.
     % be careful later, about multiple events

cbg_verb_relationship(Rel,VO,VO,CS,CS) :- !.    %continue through


cbg_other_relationship([Relation,A,B],VO,IVO, OldCS,IntCS) :-
     prep_set_correct_objs(A,B,RealA,RealB),
     spatial_defs([Relation,RealA,RealB],VO,IVO, OldCS,IntCS), !.
cbg_other_relationship([Relation,A,B],VO,IVO, OldCS,IntCS) :-
     temporal_defs([Relation,A,B],VO,IVO, OldCS,IntCS), !.

%
%    Else not defined
%
cbg_other_relationship(Rel,VO,VO,CS,CS) :- !,
     nl, write('Could Not handle -> '), write(Rel), nl,nl.
```

```
%-----------------------------------------------------
%  If a Spatial preposition relation is applying to the *event*,
%  then apply the spatial prepositon to the *actor*
%
%  Exception is instrument case, this is handled within the
%  the spatial preps
%
%-----------------------------------------------------
prep_set_correct_objs(A,B,RealA,RealB) :-
    event(Event),
    actor(Actor),
    (A = Event -> RealA = Actor; RealA = A),
    (B = Event -> RealB = Actor; RealB = B), !.
```

```
%%=============================================
%%    parself.cbm
%%
%%      Parses the CLE Logical Form Representation
%%
%%      Using BNF notation for syntax of logical form
%%      language used in the CLE.
%%      Reference: [page 12, CCSRC-005 July 1988]
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================

parse_lf(LF) :- clare_lf(LF,[]),
                write('Good Parse.').

%-------------------------------------------
% Clare's Logical Form Syntax
%-------------------------------------------

clare_lf --> s_type, body.
clare_lf --> body.                          %especially for subclauses

s_type --> [dcl].

lf_formula([LF|S],S) :- LF =.. [quant|S0], quant_1(S0,[]), !.
lf_formula([LF|S],S) :- LF =.. [quant|S0], quant_2(S0,[]), !.
lf_formula([S0|S],S) :- funct(S0,[]),
                        assertz(lfi(S0)), !.

quant_1 --> quantifier, variable, restriction, body, !.
quant_2 --> wh_sense,    variable, restriction, body, !.
funct   --> functor, argument, argumentn, !.


quantifier --> [forall], !.
quantifier --> [exists], !.
quantifier --> [most], !.

% quantifier --> variable, lambda, variable, lambda, lf_formula.
% lambda     --> ^.

wh_sense --> [wh1].
wh_sense --> [count].

% restriction(S0,S) :- [RS|Rem] = S0,
%                      ARS = [lf_object|RS],
%                      lf_formula([ARS|Rem],S).
```

```
restriction([F|S],S) :- F = [proposition,VAR,WIN2],
                            assert(lfi([proposition,VAR])),
                            assert(subwindow("'Event 2'",[WIN2])).
restriction([F|S],S) :- F = [subclause,VAR,WIN2],
                            VAR = [_SharedObject,VarShObj],
                            assert(lfi(VAR)),
                            assert(subwindow("'Relative Clause'",
                            [quant(exists,VarShObj,VAR,WIN2)])).
restriction -->  lf_formula, !.

body -->  lf_formula, !.

argument -->  term, !.
argument -->  lf_formula, !.
% argument -->  variable, lambda, lf formula.

argumentn([],[]) :- !.
argumentn -->  argument, argumentn, !.

term -->  variable, !.
term -->  constant, !.
term -->  kind, !.
term -->  special_function, !.

% kind([K|S],S) :- K =.. [kind,S0,S1], kind_a(S0,S1).
% kind_a -->  variable, restriction.

kind([K|S],S) :- K =.. [kind,V,R],
                        variable([V|[]],[]),
                        restriction([R|[]],[]), !.


% Assign variables to a letter so the structure
% remains after being assigned to the data base
variable([S0|S],S) :- var(S0), gensym(obj,Sym), S0 = Sym, !.
variable([S0|S],S) :- atom(S0), !.

constant([S0|S],S) :- atom(S0), !.

functor([S0|S],S) :- atom(S0), !.


%
%  Add the special function date added to Clare2
%
special_function([SF|S],S) :- SF =.. [sf|S0], sfdate(S0), !.

sfdate(D) :- sfdate2_0(D).              %for Clare 2.0
```

```
sfdate(D) :- sfdate2_5(D).                    %for Clare 2.5

sfdate2_0([D]) :- D =.. [date,_Year,_Month,_Day], !.
sfdate2_5([D]) :- D = date([_Year,_Month,_Day]), !.
```

```
%%=============================================
%% normalzc.cbm
%%
%%      Normalizes the constraints --
%%          Rewrites the constraint list in order of priority.
%%          Priority 10 is Highest
%%          Priority  0 is Lowest
%%
%%     This is needed so the CSP solver can attempt to solve
%%     the most important constraints first.
%%
%%     Definition of a legal solution is if achieves all
%%     priorites 6 and higher.  These are the relative constraints
%%     generated directly from the LF.  The general constraints
%%     from the Naive Physics, Missing Information, or Nice Pictures
%%     have priorities 0 through 5.  It is nice if they can be
%%     satisfied, but not required for a legal solution.
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================

%---------------------------------------------
% normalize_constraints/2
%
%     Input:
%         list of constraints including complex objects
%         Example:  [ left(A, B, 8),
%                     above(A, C, 10),
%                     touching(B, C, 2) ]
%
%     Output:
%         list of constraints that is in order of precedence
%         by highest priority first.
%         Example:  [ above(A, C, 10),
%                     left(A, B, 8),
%                     touching(B, C, 2) ].
%---------------------------------------------
normalize_constraints(CS,SpatialCS,TimeCS) :-

    %get the Spatial Constraints in Order
    nc_check_each_level_s(CS,0, [],SpatialCS),

    %get the Temporal Constrs have Priority 100
    nc_get_same_plist(CS,100,[],TimeCS).

nc_check_each_level_s(_CS,11,RetAllPlist,RetAllPlist).
nc_check_each_level_s(CS,Plevel,OldPlist,RetAllPlist) :-
```

```
    nc_get_same_plist(CS,Plevel,[],SamePlist), !,
    append(SamePlist,OldPlist,TotalPlist), !,
    NPlevel is Plevel + 1, !,
    nc_check_each_level_s(CS,NPlevel,TotalPlist, RetAllPlist), !.

nc_get_same_plist([],_Plevel,SameLevPlist,SameLevPlist) :- !.
nc_get_same_plist([FirstConstr|OtherConstr],Plevel,Plist,SameLevPlist) :-
     nc_get_priority(FirstConstr,Priority), !,
     nc_build_list_in_priority(FirstConstr,Priority,Plevel,Plist,NewPlist), !,
     nc_get_same_plist(OtherConstr,Plevel,NewPlist,SameLevPlist), !.

nc_get_priority(C,P):-
     C =.. [not|C2],
     nc_get_priorityN(C2,P).
nc_get_priority(C,P):-
     C =.. [ F,temporal|_],
     P = 100.
nc_get_priority(C,P):-
     C =.. [ Func,Obj1,Obj2,Priority],
     is_valid_priority(Priority),
     is_visual_object_tag(Obj1),
     is_visual_object_tag(Obj2),
     P = Priority.
nc_get_priority(C,_P):-
     write('Not a valid constraint'), write(C),nl.

nc_get_priorityN([C2],P):- !, nc_get_priority(C2,P).

nc_build_list_in_priority(C,P,Plevel,Plist,NewPlist) :-
% if Priority equal current Priority level,
% then put it on the list
    P = Plevel,
    NewPlist = [C|Plist], !.
nc_build_list_in_priority(_C,_P,_Plevel,Plist,Plist).

%-----------------------------------------------
% is a valid priority
%-----------------------------------------------
is_valid_priority(Priority) :-
    Priority >= 0,
    Priority =< 10, !.
is_valid_priority(Priority) :-
    fail,
    nl, write('Invalid Priority Value of '),
    write(Priority), nl.
```

Appendix C

Pictorial Primitives

```
%%=============================================
%% sprimitv.cbm
%%
%%      Set of spatial primitives to be used
%%      by the constraint builder
%%
%% Jul 90     - orginal set of primitives
%% Sep 90     - handle group of objects (of UX,LX,UY,LY),
%%              not just primitives on one object
%%              Also to handle Size
%% Jan 91     - add another parameter to Visual Object Record
%% Oct 91     - new version of touching, overlap, near, and far
%% May 92     - rewritten in linear, conjunctive constraints
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================

%---------------------------------------------
% CONSTRAINTSt Defintion for Object Groups
%     an object group is made up of one or more
%     visual objects.
%     Variables:
%          VOG -- visual object group ID
%          W   -- actual word
%          R   -- upper X value (right)
%          L   -- lower X value (left)
%          T   -- upper Y value (top)
%          B   -- lower Y value (bottom)
%
% (all relation names are infix operation).
%---------------------------------------------

%---------------------------------------------
```

```
% ABOVE, BELOW, RIGHT, LEFT
%
% Definitive relations between two visual object groups
%-----------------------------------------------------
above([_,_,L1,_R1, B1,_T1], [_,_,L2,_R2,_B2, T2],_P) :- B1 #>= T2.
below([_,_,L1,_R1,_B1, T1], [_,_,L2,_R2, B2,_T2],_P) :- T1 #<= B2.
right([_,_, L1,_R1,_B1,_T1], [_,_,L2, R2,_B2,_T2],_P) :- L1 #>= R2.
left( [_,_,L1, R1,_B1,_T1], [_,_, L2,_R2,_B2,_T2],_P) :- R1 #<= L2.

%-----------------------------------------------------
% NOT ABOVE, NOT BELOW, NOT RIGHT, NOT LEFT
% % Definitive relations between two visual object groups
%-----------------------------------------------------
not_above([_,_,_L1,_R1, B1,_T1], [_,_,_L2,_R2, B2, T2],_P)  :- B1 #<= T2.
not_below([_,_,_L1,_R1,_B1, T1], [_,_,_L2,_R2, B2,_T2],_P)  :- T1 #>= B2.
not_right([_,_, L1,_R1,_B1,_T1], [_,_,_L2, R2,_B2,_T2],_P)  :- L1 #<= R2.
not_left( [_,_,_L1, R1,_B1,_T1], [_,_, L2,_R2,_B2, T2], P)  :- R1 #>= L2.

%-----------------------------------------------------
% IN-LEFT-THIRD,IN-MIDDLE-HORIZ,IN-RIGHT-THIRD
%
% the epicenter of object 1 is constrained within the region of 2
%-----------------------------------------------------
in_left_third([_,Tag1, L1,_R1,_B1,_T1], [_,Tag2, L2,_R2,_B2,_T2],_P) :-
    bil_get_icon_data(Tag1,_NN1,Width1,_Height1,_IconPic1),
    bil_get_icon_data(Tag2,_NN2,Width2,_Height2,_IconPic2),
    HalfLengthX_of_1 is Width1 // 2,
    ThirdLengthX_of_2 is Width2 // 3,

% L1 + HalfLengthX_of_1 #>= L2,
% L1 + HalfLengthX_of_1 #<= L2 + ThirdLengthX_of_2,
% L1 #>= L2 + ConstA,
% L1 #<= L2 + ConstB,
    ConstA is 0 - HalfLengthX_of_1,
    ConstB is ThirdLengthX_of_2 - HalfLengthX_of_1,

    %constraints are:
    L1 #>= L2 + ConstA,
    L1 #<= L2 + ConstB.


in_right third([_,Tag1,L1,_R1,_B1, T1], [_,Tag2,_L2, R2,_B2,_T2],_P) :-
    bil_get_icon_data(Tag1,_NN1,Width1,_Height1,_IconPic1),
    bil_get_icon_data(Tag2,_NN2,Width2,_Height2,_IconPic2),
    HalfLengthX_of_1 is Width1 // 2,
    ThirdLengthX_of_2 is Width2 // 3,

% L1 + HalfLengthX_of_1 #>= R2 - ThirdLengthX_of_2,
```

```
%  L1 + HalfLengthX_of_1 #<= R2
%  L1 #>= R2 + ConstA,
%  L1 #<= R2 + ConstB,
   ConstA1 is 0 - ThirdLengthX_of_2,
   ConstA  is ConstA1 - HalfLengthX_of_1,
   ConstB is 0 - HalfLengthX_of_1,

   %constraints are:
   L1 #>= R2 + ConstA,
   L1 #<= R2 + ConstB.


in_middle_horiz([_,Tag1, L1,_R1,_B1,_T1], [_,Tag2, L2, R2,_B2,_T2],_P) :-
   bil_get_icon_data(Tag1,_NN1,Width1,_Height1,_IconPic1),
   bil_get_icon_data(Tag2,_NN2,Width2,_Height2,_IconPic2),
   HalfLengthX_of_1 is Width1 // 2,
   ThirdLengthX_of_2 is Width2 // 3,

%  L1 + HalfLengthX_of_1 #>= L2 + ThirdLengthX_of_2,
%  L1 + HalfLengthX_of_1 #<= R2 - ThirdLengthX_of_2,
%  L1 #>= L2 + ConstA,
%  L1 #<= R2 + ConstB,
   ConstA  is ThirdLengthX_of_2 - HalfLengthX_of_1,
   ConstB1 is 0 - ThirdLengthX_of_2,
   ConstB  is ConstB1 - HalfLengthX_of_1,

   %constraints are:
   L1 #>= L2 + ConstA,
   L1 #<= R2 + ConstB.


%-----------------------------------------------
%  IN-TOP-THIRD,IN-MIDDLE-VERT,IN-BOTTOM-THIRD
%
% the epicenter of object A is placed within the region of B
%-----------------------------------------------
in_bottom_third([_,Tag1,_L1,_R1, B1,_T1], [_,Tag2,_L2,_R2, B2,_T2],_P) :-
   bil_get_icon_data(Tag1,_NN1,_Width1,Height1,_IconPic1),
   bil_get_icon_data(Tag2,_NN2,_Width2,Height2,_IconPic2),
   HalfLengthY_of_1 is Height1 // 2,
   ThirdLengthY_of_2 is Height2 // 3,

%  B1 + HalfLengthY_of_1 #>= B2,
%  B1 + HalfLengthY_of_1 #<= B2 + ThirdLengthY_of_2,
%  B1 #>= B2 + ConstA,
%  B1 #<= B2 + ConstB,
   ConstA is 0 - HalfLengthY_of_1,
   ConstB is ThirdLengthY_of_2 - HalfLengthY_of_1,
```

```
    %constraints are:
    B1 #>= B2 + ConstA,
    B1 #<= B2 + ConstB.


in_top_third([_,Tag1,_L1,_R1, B1,_T1], [_,Tag2,_L2,_R2,_B2, T2],_P) :-
    bil_get_icon_data(Tag1,_NN1,_Width1,Height1,_IconPic1),
    bil_get_icon_data(Tag2,_NN2,_Width2,Height2,_IconPic2),
    HalfLengthY_of_1 is Height1 // 2,
    ThirdLengthY_of_2 is Height2 // 3,

%   B1 + HalfLengthY_of_1 #>= T2 - ThirdLengthY_of_2,
%   B1 + HalfLengthY_of_1 #<= T2
%   B1 #>= T2 + ConstA,
%   B1 #<= T2 + ConstB,
    ConstA1 is 0 - ThirdLengthY_of_2,
    ConstA  is ConstA1 - HalfLengthY_of_1,
    ConstB is 0 - HalfLengthY_of_1,

    %constraints are:
    B1 #>= T2 + ConstA,
    B1 #<= T2 + ConstB.

in_middle_vert([_,Tag1,_L1,_R1, B1,_T1], [_,Tag2,_L2,_R2, B2, T2],_P) :-
    bil_get_icon_data(Tag1,_NN1,_Width1,Height1,_IconPic1),
    bil_get_icon_data(Tag2,_NN2,_Width2,Height2,_IconPic2),
    HalfLengthY_of_1 is Height1 // 2,
    ThirdLengthY_of_2 is Height2 // 3,

%   B1 + HalfLengthY_of_1 #>= B2 + ThirdLengthY_of_2,
%   B1 + HalfLengthY_of_1 #<= T2 - ThirdLengthY_of_2,
%   B1 #>= B2 + ConstA,
%   B1 #<= T2 + ConstB,
    ConstA  is ThirdLengthY_of_2 - HalfLengthY_of_1,
    ConstB1 is 0 - ThirdLengthY_of_2,
    ConstB  is ConstB1 - HalfLengthY_of_1,

    %constraints are:
    B1 #>= B2 + ConstA,
    B1 #<= T2 + ConstB.


%-----------------------------------------------
% NEAR
% Conjunctive description that maps a square region surrounding
% of object 1, by distance of "near" distance plus diameter of object 2
```

```
%
% This is considered that an object will be checked for overlap later.
% The overlap check is disjunctive, and will be applied after all
% conjunctive constraints are applied.
%-----------------------------------------------------------
near(VO1,VO2,_P) :-
    get_near_value(D),
    calculate_diameter(VO2,DiamX2,DiamY2),
    calculate_ok_region(DiamX2, DiamY2, D, DistX, DistY),
    constrain_within_region(VO1,VO2,DistX,DistY).

calculate_epicenter([_VO,_,L,R,B,T],ECX,ECY) :-
    calculate_radius([_VO,_,L,R,B,T],RadX,RadY),
    ECX is L + RadX,
    ECY is B + RadY.

calculate_radius([_VO,_,L,R,B,T],RadX,RadY) :-
    calculate_diameter([_VO,_,L,R,B,T],DiamX,DiamY),
    RadX  is DiamX // 2,
    RadY  is DiamY // 2.

calculate_diameter([_VO,_,L,R,B,T],DiamX,DiamY) :-
    DiamX is R - L,
    DiamY is T - B.

get_near_value(D) :- near_value(D).

calculate_ok_region(DiamX2, DiamY2, D, DistX, DistY) :-
    DistX is D + DiamX2,
    DistY is D + DiamY2.

constrain_within_region([_VO1,_,L1,R1,B1,T1],[_VO2,_,L2,R2,B2,T2],DistX,DistY) :-
%   L2 #>= L1 - DistX,      %bug in CHIP, won't accept this
    L2 + DistX #>= L1,
    R2 #<= R1 + DistX,
%   B2 #>= B1 - DistY,      %bug in CHIP, won't accept this
    B2 + DistY #>= B1,
    T2 #<= T1 + DistY.


%-----------------------------------------------------------
% TOUCHING
%
% Conjunctive description of near that blocks out a region of
% the obj2 touching all sides of object1. This will be the same
% as near, with the "near" distance of zero.
%
% This is also assumes that an overlap check will later force object
```

```
% to some position in the region that is not overlap.
%-----------------------------------------------------
touching(VO1,VO2,_P) :-
    % set near value to 0
    D = 0,
    calculate_diameter(VO2,DiamX2,DiamY2),
    calculate_ok_region(DiamX2, DiamY2, D, DistX, DistY),
    constrain_within_region(VO1,VO2,DistX,DistY).


%-----------------------------------------------------
% FAR-FROM
% Conjunctive description that maps a square region surrounding
% of object 1, by distance of "far" distance plus diameter of object 2
%
% This constraint only works in the X value.
%
% Similar to near, but uses the "far_from" value, and
% constrains outside the region, not inside region.
%-----------------------------------------------------
farfrom(VO1,VO2,_P) :-
    get_far_from_value(D),
    constrain_outside_region(VO1,VO2,D).

get_far_from_value(D) :- far_from_value(D).

constrain_outside_region([_VO1,_,L1,R1,B1,T1],[_VO2,_,L2,R2,B2,T2],D) :-
    L2 #>= R1 + D, !.
constrain_outside_region([_VO1,_,L1,R1,B1,T1],[_VO2,_,L2,R2,B2,T2],D) :-
%     R2 #<= L1 - D, !.   %bug in CHIP, won't accept this
    R2 + D #<= L1, !.


%-----------------------------------------------------
% NOT OVERLAP
%
%   Important Note: That this is constraining object B,
%   and does not constrain object A.
%
%   %potential bug could be that objects that are "fixed" position
%   %may cause no solution to be found if that "fixed" object is
%   %constrained to a "floating" object
%
%   %also this constraint is described DISJUNCTIVELY. Therefore,
%   %to aid in visability of constrains being applied, it is easier
%   %to see if conjunctive constraints are applied firts, and
%   %disjuncitve at the end.
%-----------------------------------------------------
```

```
not_overlap(A,B,_P) :-
    not_overlap_constraint(A,B).

%Object B must be Below A, or Above A, or Leftof A or Rightof A.
% apply one first and see.

%below
not_overlap_constraint([_Obj1Num,_Obj1Tag,_L1,_R1, B1,_T1],
                       [_Obj2Num,_Obj2Tag,_L2,_R2,_B2, T2]) :- T2 #<= B1.
%above
not_overlap_constraint([_Obj1Num,_Obj1Tag,_L1,_R1,_B1, T1],
                       [_Obj2Num,_Obj2Tag,_L2,_R2, B2,_T2]) :- B2 #>= T1.
%left
not_overlap_constraint([_Obj1Num,_Obj1Tag, L1,_R1,_B1,_T1],
                       [_Obj2Num,_Obj2Tag,_L2, R2,_B2,_T2]) :- R2 #<= L1.
%right
not_overlap_constraint([_Obj1Num,_Obj1Tag,_L1, R1,_B1,_T1],
                       [_Obj2Num,_Obj2Tag, L2,_R2,_B2,_T2]) :- L2 #>= R1.
```

```
%%=============================================
%%  tprimitiv.cbm
%%
%%   Temporal Expressions Primitives
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================


%-------------------------------------------
%  DETERMINE TIME FUNCTIONS
%-------------------------------------------
%version Clare 2.5
determine_time_scale(SFDate,TimeScaleType):-
        SFDate = sf(date([Y,M,D])),
        TimeScaleType = 'days_of_week'.

%version Clare 2.0
determine_time_scale(SFDate,TimeScaleType):-
        SFDate = sf(date(Y,M,D)),
        TimeScaleType = 'days_of_week'.

%version Clare 2.5
determine_event_time_absolute(SFDate,Date) :-
        SFDate = sf(date(Date)).

%version Clare 2.0
determine_event_time_absolute(SFDate,Date) :-
        SFDate = sf(date(Y,M,D)),
        Date = [Y,M,D].
```

Appendix D

Verb Definitions

```
%%=============================================
%%  verb.cbm
%%
%%    Verb Pictorial Definitions
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================

%-----------------------------------------------
%  SAY                          SAY_STATE_THAT
%-----------------------------------------------
verb_defs(['say_StateThat',Event,Actor_Data,Proposition],VO,NVO,OldCS,NewCS,Actor) :-

    priority_level('actor-left-event',P_ALE),
    priority_level('actor-near-event',P_ANE),
    priority_level('event-left-patient',P_ELP),
    priority_level('prep-not-above',P_NA),
    priority_level('prep-not-below',P_NB),
    priority_level('event-farfrom-patient',P_EFFP),

    vo_modify_name_replace(Event,'say_StateThat',VO,IVO),

    (is visual_object_tag(Actor_Data) -> Actor = Actor_Data, NVO = IVO;
    vo_add_object(Actor_Data,Actor,IVO,NVO) ),

    NewCS = [
              left(Actor,Event,P_ALE),
              touching(Actor,Event,P_ANE),
              in_top_third(Event,Actor,5),
%               touching(Proposition,Event,P_ELP),
              in_middle_horiz(Proposition,Event,5),
              in_middle_vert(Proposition,Event,5)
```

```
            | OldCS], !.

%-----------------------------------------------------
%  SEE                  SEE LOOK AT
%-----------------------------------------------------
verb_defs(['see_LookAt',Event,Actor_Data,Patient],VO,NVO,OldCS,NewCS,Actor) :-

    priority_level('actor-left-event',P_ALE),
    priority_level('actor-near-event',P_ANE),
    priority_level('event-left-patient',P_ELP),
    priority_level('prep-not-above',P_NA),
    priority_level('prep-not-below',P_NB),
    priority_level('event-farfrom-patient',P_EFFP),

    vo_modify_name_concat(Patient,'_Obj',VO,IVO1),
    vo_modify_name_replace(Event,'see_LookAt',IVO1,IVO),

    (is visual_object_tag(Actor_Data) -> Actor = Actor_Data, NVO = IVO;
    vo_add_object(Actor_Data,Actor,IVO,NVO) ),

    NewCS = [
            left(Actor,Event,P_ALE),
            touching(Actor,Event,P_ANE),
            in_top_third(Event,Actor,5),
            left(Event,Patient,P_ELP),
            farfrom(Event,Patient,P_EFFP)
            | OldCS], !.

%-----------------------------------------------------
%  DRIVE                OPERATE A VEHICLE
%-----------------------------------------------------
verb_defs(['drive_CauseToOperate',Event,Actor_Data,Patient],
                                    VO,NVO,OldCS,NewCS,Actor) :-

    priority_level('actor-left-event',P_ALE),
    priority_level('actor-near-event',P_ANE),
    priority_level('event-left-patient',P_ELP),
    priority_level('prep-not-above',P_NA),
    priority_level('prep-not-below',P_NB),
    priority_level('event-farfrom-patient',P_EFFP),

    vo_modify_name_concat(Patient,'_Obj',VO,IVO1),
    vo_modify_name_replace(Event,'drive_CauseToOperate',IVO1,IVO),

    (is visual_object_tag(Actor_Data) -> Actor = Actor_Data, NVO = IVO;
    vo_add_object(Actor_Data,Actor,IVO,NVO) ),

    NewCS = [
```

```
left(Actor,Event,P_ALE),
touching(Actor,Event,P_ANE),
in_middle_vert(Event,Actor,5),
left(Event,Patient,P_ELP),
near(Event,Patient,P_EFFP)
| OldCS], !.
```

Appendix E

Preposition Definitions

```
%%=============================================
%%  spatial.cbm
%%
%%   Spatial Preposition Pictorial Definitions
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================


%-----------------------------------------------
%  ON                    ON CONCERNING
%-----------------------------------------------
spatial_defs(['on_Concerning',A,B],VO,NVO,OldCS,NewCS) :-
    NewCS = [
              left(A,B,10),
              touching(A,B,10)
              | OldCS],
    vo_modify_name_concat(A,'*',VO,IVO),
    reverse(IVO,NVO), !.

%-----------------------------------------------
%  ON                    LOCATIONAL
%-----------------------------------------------
spatial_defs(['on_Locational',A,B],VO,VO,OldCS,NewCS) :-

priority_level('prep-touching',P_T),
priority_level('prep-not-left',P_NL),
priority_level('prep-not-right',P_NR),
priority_level('prep-above',P_A),
priority_level('prep-middle-horiz',P_MidH),

    NewCS = [
              above(A,B,P_A),
```

```
            not(left(A,B,P_NL)),
            not(right(A,B,P_NR)),
            in_middle_horiz(A,B,P_MidH),
            touching(A,B,P_T)
            | OldCS], !.

%-----------------------------------------------------
%  ON                  SPATIALLY ON TOP OF
%-----------------------------------------------------
spatial_defs(['on_SpatiallyOnTopOf',A,B],VO,VO,OldCS,NewCS) :-

   NewCS = [
            above(A,B,10),
            not(left(A,B,10)),
            not(right(A,B,10)),
            touching(A,B,10)
            | OldCS], !.

%-----------------------------------------------------
%  ON                  generic ON
%-----------------------------------------------------
spatial_defs(['on',A,B],VO,VO,OldCS,NewCS) :-

   NewCS = [
            above(A,B,10),
            not(left(A,B,10)),
            not(right(A,B,10)),
            touching(A,B,10)
            | OldCS], !.

%-----------------------------------------------------
%  WITH                WITH ACCOMPANYING
%-----------------------------------------------------
spatial_defs(['with_Accompanying',A,B],VO,VO,OldCS,NewCS) :-

priority_level('prep-near',P_N),
priority_level('prep-not-above',P_NA),
priority_level('prep-not-below',P_NB),
priority_level('prep-left',P_L),

   NewCS = [
            touching(B,A,P_N),
            not(above(B,A,P_NA)),
            not(below(B,A,P_NB)),
            left(B,A,P_L)
            | OldCS], !.

%-----------------------------------------------------
```

```
% WITH                    WITH HAVING
%-----------------------------------------------------
spatial_defs(['with_Having',A,B],VO,VO,OldCS,NewCS) :-

    NewCS = [
              near(A,B,6),
              not(above(A,B,8)),
              not(below(A,B,8))
              | OldCS], !.

%-----------------------------------------------------
% WITH                    WITH INSTRUMENT
%-----------------------------------------------------
spatial_defs(['with_Instrument',A,B],VO,NVO,OldCS,NewCS) :-

priority_level('prep-left',P_L),
priority_level('prep-touching',P_T),
priority_level('prep-not-above',P_NA),
priority_level('prep-not-below',P_NB),
priority_level('prep-middle-vert',P_MidV),

%if instrument attach to event, not actor
      event(Event),actor(Actor),
     (A = Actor -> E = Event; E = A),

    NewCS = [
              left(E,B,P_L),
              touching(E,B,P_T),
              in_middle_vert(E,B,P_MidV)
              | OldCS],
    vo_modify_name_concat(B,'_Instr',VO,NVO), !.

%-----------------------------------------------------
% WITH                    WITH USING
%-----------------------------------------------------
spatial_defs(['with_Using',A,B],VO,NVO,OldCS,NewCS) :-

priority_level('prep-left',P_L),
priority_level('prep-touching',P_T),
priority_level('prep-not-above',P_NA),
priority_level('prep-not-below',P_NB),
priority_level('prep-middle-vert',P_MidV),

%if instrument attach to event, not actor
      event(Event),actor(Actor),
     (A = Actor -> E = Event; E = A),

    NewCS = [
```

```
                left(E,B,P_L),
                touching(E,B,P_T),
                in_middle_vert(E,B,P_MidV)
                | OldCS],
    vo_modify_name_concat(B,'_Instr',VO,NVO), !.

%-----------------------------------------------------
%  WITH                GENERIC
%-----------------------------------------------------
spatial_defs(['with',A,B],VO,VO,OldCS,NewCS) :-
    NewCS = [
                near(A,B,6),
                not(above(A,B,8)),
                not(below(A,B,8))
                | OldCS], !.
```

```
%%=============================================
%%  temporal.cbm
%%
%%    Temporal Expressions Pictorial Definitions
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================

%---------------------------------------------
%  CURRENT_TIME                    CURRENT TIME MARK
%---------------------------------------------
%temporal_defs(['current_time',CT_Obj],VO,NVO,OldCS,NewCS) :-
%     NewCS = [
%               time_set_current_time(temporal,'nnooww')
%             | OldCS], !.

%---------------------------------------------
%  ON_TEMPORAL                     ON A DATE
%---------------------------------------------
temporal_defs(['on_Temporal',Event,SFDate],VO,VO,OldCS,NewCS) :-

    determine_time_scale(SFDate,ScaleType),
    determine_event_time_absolute(SFDate,Date),
    NewCS = [
              plot_time_and_scale(temporal,Date,ScaleType)
              | OldCS], !.

%---------------------------------------------
%  PRECEDES_IN_TIME                PAST
%---------------------------------------------
temporal_defs(['precedes_in_time',Event,CurrentTime],VO,NVO,OldCS,NewCS) :-

% current time is *not* an object in Spatial PRW, so remove

    is_visual_object_tag(CurrentTime),
    vo_remove_object(CurrentTime,VO,NVO),

    NewCS = [
              plot_time_and_scale(temporal,'past','past_now_future')
              | OldCS], !.

%---------------------------------------------
%  PRECEDES_IN_TIME (*FOR CLARE2.0*)      PAST
%---------------------------------------------
temporal_defs(['precedes_in_time',Event,CurrentTime],VO,VO,OldCS,NewCS) :-

    %this is for runs from Clare2.0
```

```
%this date is not of much value, it contains the date
%that the run was made.

%so just call it past tense

NewCS = [
        plot_time_and_scale(temporal,'past','past_now_future')
        | OldCS], !.
```

Appendix F

Non–LF Constraint Modules

```
%%=============================================
%% nphysics.cbm
%%      A simplistic module that contains
%%      naive physics constraints, such as gravity.
%%
%% defintion of constraint strucutures
%%
%%      OPOL – old pictorial object list
%%      NPOL – new pictorial object list
%%      OCS  – old constraint structure
%%      NCS  – new constraint structure
%%      NPCS – naive physics added constraint structure
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================

naive_physics_constraints(POL) :-
   npcm_touch_the_ground(POL).

%---------------------------------------------
% create an object called the ground
%---------------------------------------------
%   NPOL = [pictorial_object([obj0,'pictorial_Ground',_X,_UX,_Y,_UY])
%            | OPOL],

%---------------------------------------------
% check to see if an object "could" (legally) touch the ground.
% (that the Bottom variable can accept a value of 0).
%
% If so make a constraint touching(A,ground)
%---------------------------------------------
npcm_touch_the_ground([]).
npcm_touch_the_ground([ObjF|ObjN]) :-
```

```
    npcm_gravity_constraint(ObjF),
    npcm_touch_the ground(ObjN).

npcm_gravity_constraint(ObjF) :-
    write('ObjF is'),write(ObjF),
    ObjF = pictorial_object([_ObjID,ObjTag,_L,_R,B,_T]),
    nl,nl,write('Try to apply Gravity to obj '), write(ObjTag),
    nl, write('Bottom Variable is '),write(B),
    B #= 0, write('Gravity has an Effect'), nl, !.
npcm_gravity_constraint(ObjF) :-
    ObjF = pictorial_object([_ObjID,ObjTag,_L,_R,_B, T]),
    nl,write('Gravity does NOT effect object '),write(ObjTag),nl, !.
```

```
%%=============================================
%% nicepict.cbm
%%
%%      A "nice pictures" constraint module
%%
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================
nice_pict_constraints(POL).

%  kill_overlap(VisObjs,OldCS,NewCS).

%-------------------------------------------
% Centering  of overall picture
%     --center overall picture on the grid
%-------------------------------------------

%-------------------------------------------
% Margin
%     --forced margin or a different type of overall centering
%-------------------------------------------


%-------------------------------------------
%  KILL OVERLAP
%-------------------------------------------
kill_overlap(VO,LF,FLF) :-
        trav_vol(VO,LF,FLF).
trav_vol([],ILF,ILF).
trav_vol([A|Rem],LF,FLF) :-
     pictorial_object([Obj1,_,_,_,_]) = A,
     trav_vol2(Obj1,Rem,LF,ILF),
     trav_vol(Rem,ILF,FLF).
trav_vol2(_Obj1,[],ILF,ILF).
trav_vol2(Obj1,[B|Rem],LF,FLF):-
     pictorial_object([Obj2,_,_,_,_]) = B,
     priority_level('not-overlap',PLevel),
     ILF = [not(overlap(Obj1,Obj2,PLevel))|LF],
     trav_vol2(Obj1,Rem,ILF,FLF).
```

```
%%=============================================
%% missinfo.cbm
%%
%%      Missing Information Module
%%
%%      This module narrows the constraints further if needed.
%%      Often, the solution is a ragne of soltions, and to draw
%%      the picture an exact placement is needed.
%%
%%      These Missing-Info Constraints are "weak" in the sense, that
%%      they only apply if *after* the other constraints are applied
%%      that the solution is still to large.
%%
%%      For example, a cup is on the table, but there isn't any
%%      additional information supplying the location of the cup
%%      on the table.
%%
%%      A problem is that pictorial representations can be created
%%      that yeild false implicatures.  Care must be taken to choose
%%      missing-info constraints in a way not to display the information
%%      so that a false implicature is caused.  This module attempts
%%      to minimize this problem.
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================

missing_information_constraints(POL) :-

%
% center glass on table by not left 1/3, not right 1/3
%
   hard_constrain_objects(POL).

%---------------------------------------------
%       hard_constrain_one_object(POL)
%
%       If all the objects are "loose" on the grid, a "solid"
%       picture may be formed if one object is placed exactly.
%       It is placed within the epicenter of its legal limits.
%---------------------------------------------
hard_constrain_objects([]).
hard_constrain_objects([Obj1|RemObjs]) :-
   hard_constrain_one_object(Obj1),
hard_constrain_objects(RemObjs).

hard_constrain_one_object(Obj1) :-
   Obj1 = pictorial_object([ObjNum,ObjID,L,R,B,T]),
```

```
%calculate epicenter of solution region
maxdomain(L,SR_MaxL),
maxdomain(B,SR_MaxB),
mindomain(L,SR_MinL),
mindomain(B,SR_MinB),
SR_Mid1L is SR_MaxL - SR_MinL,
SR_Mid1B is SR_MaxB - SR_MinB,
SR_Mid2L  is SR_Mid1L // 2,
SR_Mid2B  is SR_Mid1B // 2,
SR_MidL  is SR_Mid2L + SR_MinL,
SR_MidB  is SR_Mid2B + SR_MinB,

%constrain the left bottom corner of icon
% to epicenter of soltion region
L #= SR_MidL,
B #= SR_MidB.
```

Appendix G

Other Constraint Building Routines

```
%%=============================================
%%    dstructs.cbm
%%
%%    The data structures handling routines
%%
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================

%-------------------------------------------------
%     get_all_variables(Objects,VARS)
%
%       Objects is a list of pictorial Objects
%       VARS is a list of *all* of the variables associated with
%          an object.  Each pictorial object has a four variables:
%          top, bottom,left, and right positions on the
%          two-dimensional grid.
%-------------------------------------------------
get_all_variables(Objects,VARS) :-
    get_all_variables(Objects,[],VARS).
get_all_variables([],VARS,VARS).
get_all_variables([Obj1|RemObjs],CurrVars,VARS) :-
    get_vars_from_object(Obj1,_ID,_Tag,L,R,B,T),
    NewVars = [L,R,B,T| CurrVars],
    get_all_variables(RemObjs,NewVars,VARS).

%-------------------------------------------------
%  get_expanded_list_Constr(Constrs,Objects,ExpC),
%
%     Make a list of FULL Expanded Constraints, replacing
```

```
%       the tag Object with the full object record
%-----------------------------------------------------------
get_expanded_list_Constrs(Constrs,Objects,ExpCL) :-
    get_expanded_list_Constrs(Constrs,Objects,[],ExpCL).

get_expanded_list_Constrs([],_Objects,ExpCL,ExpCL).
get_expanded_list_Constrs([Constr1|RemConstr],Objects,CurrCL,TotCL) :-
    get_expanded_list_Constrs(RemConstr,Objects,CurrCL,IntCL),
    expand_constraint(Constr1,Objects,ExpConstr1),
    TotCL = [ExpConstr1|IntCL].

%-----------------------------------------------------------
% expand the constraint structure
%     call by:
%          expand_constraint(Constr,CurrObjectsPos,ExpConstr)
%-----------------------------------------------------------
expand_constraint(NotConstr,CurrObjectsPos,NotExpConstr) :-
     NotConstr =.. [not,Constr],
     Constr    =.. [Rel,Obj1,Obj2,Pr], !,
     build_expanded_ds(Obj1,CurrObjectsPos,ExpObj1),
     build_expanded_ds(Obj2,CurrObjectsPos,ExpObj2),
     ExpConstr    =.. [Rel,ExpObj1,ExpObj2,Pr],
     NotExpConstr =.. [not,ExpConstr].

expand_constraint(Constr,CurrObjectsPos,ExpConstr) :-
     Constr    =.. [Rel,Obj1,Obj2,Pr], !,
     build_expanded_ds(Obj1,CurrObjectsPos,ExpObj1),
     build_expanded_ds(Obj2,CurrObjectsPos,ExpObj2),
     ExpConstr    =.. [Rel,ExpObj1,ExpObj2,Pr].

build_expanded_ds(Obj,[VObj|_VObjRem],ExpObj) :-
     VObj = pictorial_object([Obj,W,L,R,B,T]),
     ExpObj = [Obj,W,L,R,B,T], !.
build_expanded_ds(Obj,[_VObj|VObjRem],ExpObj) :-
     build_expanded_ds(Obj,VObjRem,ExpObj).

%-----------------------------------------------------------
%  get the vars from one object
%-----------------------------------------------------------
get_vars_from_object(ObjectRecord,ID,Tag,L,R,B,T) :-
  ObjectRecord = pictorial_object([ID,Tag,L,R,B,T]).

%-----------------------------------------------------------
% it is a visual object tag
%-----------------------------------------------------------
is_visual_object_tag(A) :-
    atom(A),
    string_search(1,'obj',A,0).
```

```
%-----------------------------------------------
% Write out the Solution from its list format
%     --an alternative to displaying the result
%-----------------------------------------------
s_write([]) :- nl.
s_write([S1|S]) :-
        write(S1),nl,
        s_write(S).

sr_write([]) :- nl,nl,nl.
sr_write([CS1|CS]) :-
        CS1 =.. [Func,Obj1,Obj2,Priority],
        get_vars_from_object(Obj1,W1,_L1,_R1,_B1,_T1),
        get_vars_from_object(Obj2,W2,_L2,_R2,_B2,_T2),
        write(Func),write('('),write(W1),write(','),write(W2),write(')'),nl,
        sr_write(CS).

%-----------------------------------------------
% Object on the Grid ?
%-----------------------------------------------
object_on_grid([_Object,X,Y]) :-
        X >= 0,
        grid_size_x(XLIM),
        X =< XLIM,
        Y >= 0,
        grid_size_y(YLIM),
        Y =< YLIM.

%-----------------------------------------------
%   Pictorial Object Modifications
%-----------------------------------------------
vo_modify_name_concat(ObjID,Mod_add,OldVOL,NewVOL) :-
    build_expanded_ds(ObjID,OldVOL,[ObjID,Word,A,B,C,D]),
    rem_vo_in_vol([ObjID,Word,A,B,C,D],OldVOL,IntVOL),
    concat(Word,Mod_add,NewWord),
    NewVOL = [pictorial_object([ObjID,NewWord,A,B,C,D])|IntVOL].

vo_modify_name_replace(ObjID,Mod_rep,OldVOL,NewVOL) :-
    build_expanded_ds(ObjID,OldVOL,[ObjID,Word,A,B,C,D]),
    rem_vo_in_vol([ObjID,Word,A,B,C,D],OldVOL,IntVOL),
    NewVOL = [pictorial_object([ObjID,Mod_rep,A,B,C,D])|IntVOL].

rem_vo_in_vol(Obj,OV,NV) :-
   travers_vol(Obj,OV,[],NV).
travers_vol(_Obj,[],FL,FL) :- !.
travers_vol(Obj,[Obj1|ObjRem],BL,FL) :-
   pictorial_object(Obj) = Obj1,
```

```
    travers_vol(Obj,ObjRem,BL,FL), !.
travers_vol(Obj,[Obj1|ObjRem],BL,FL) :-
    travers_vol(Obj,ObjRem,[Obj1|BL],FL), !.

vo_add_object(New_Obj Data,New_Obj_Tag,OldVOL,NewVOL) :-
     gensym(obj,New_Obj_Tag),
     mvo([New_Obj_Data,New_Obj_Tag],OldVOL,NewVOL).

vo_remove_object(ObjID,OldVOL,NewVOL) :-
     build_expanded_ds(ObjID,OldVOL,[ObjID,Word,A,B,C,D]),
     rem_vo_in_vol([ObjID,Word,A,B,C,D],OldVOL,NewVOL).

%-----------------------------------------------
%   Calculate Epicenter, Diameter, Radius of an Object
%-----------------------------------------------
calculate_relative_epicenter([_ObjID,ObjTag,L,_R,B,_T],ECX,ECY) :-
     calculate_radius([_ObjID,ObjTag,_L,_R,_B,_T],RadX,RadY),
     ECX is L + RadX,
     ECY is B + RadY.

calculate_radius(PictObj,RadX,RadY) :-
     calculate_diameter(PictObj,DiamX,DiamY),
     RadX  is DiamX // 2,
     RadY  is DiamY // 2.

calculate_diameter([_ObjID,ObjTag,_L,_R,_B,_T],Width,Height) :-
     bil_get_icon_data(ObjTag,_NickName,Width,Height,_Icon).
```

```
%%=============================================
%% userdef.cbm
%%
%%      User defined variables for the Constraint Satisfaction
%%      Problem (CSP) Solver.
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================

%
% User defined variables
%

% Subjective Constraint definitions
%
near_value(5).                          % near distance
far_from_value(120).                    % far distance

% Grid sizes
%
max_grid_size_x(260).                   % max grid size in x
max_grid_size_y(140).                   % max grid size in y

%
%  Global Priority Levels

priority_level('actor-left-event',9).
priority_level('actor-near-event',10).
priority_level('event-left-patient',9).
priority_level('event-farfrom-patient',9).

priority_level('prep-touching',8).
priority_level('prep-left',7).
priority_level('prep-right',7).
priority_level('prep-above',8).
priority_level('prep-below',8).
priority_level('prep-not-left',7).
priority_level('prep-not-right',7).
priority_level('prep-not-above',7).
priority_level('prep-not-below',7).
priority_level('prep-near',7).

priority_level('prep-middle-horiz',5).
priority_level('prep-middle-vert',5).

priority_level('not-overlap',4).
```

```
priority_level('gravity-not-below',2).
priority_level('gravity-touch',1).
```

```
%%================================================
%% example.lfs
%%
%%      Loads the LFs into the system
%%     for the TTPS
%%
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%================================================


% Load the LF files
%
:- ['INPUT/cmht.lfs'].          % Carla, Man, Hill, Telescope, 7LFs
%:- ['INPUT/ashswot.lfs'].      % Alan said he saw the woman on Tuesday, 4LFs
%:- ['INPUT/mwdcscot.lfs'].     % Man,who drove car saw cat on table, 2LFs
```

```
%%=============================================
%% captsols.csp
%%
%%     Captures all of the solutions for the multiple LFs
%%     and writes them to a file to be used for graphics
%%     modules on another system.
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================
capture_solutions(OUTPUT,Sentence) :-

    concat('OUTPUT/',OUTPUT,OUTF),
    open(OUTF,write,s),

    write('solutionInfo(',s),
    write("'",s),
    write(Sentence,s),
    write("'",s),
    write(',',s),
    write_each_lf_solution(s),
    write(' ). ',s),
    close(s).

write_each_lf_solution(S) :-
        retract(pr_solution(LF_Num,LF_Solution)),
        write('[',S),
        write(pr_sol(LF_Num,LF_Solution),S),
write_next_lf_solutions(S).

write_next_lf_solutions(S) :-
        retract(pr_solution(LF_Num,LF_Solution)), !,
        write(',',S),
        write(pr_sol(LF_Num,LF_Solution),S),
        write_next_lf_solutions(S).

write_next_lf_solutions(S) :-
        write(']',S).
```

Appendix H

CSP Solver

```
%%=============================================
%% cspsolve2.csp
%%
%%     Main file for CSP Solver
%%          Routine assigns a domain to each object
%%          and then applies each constraint by
%%          making calls to CHIP (Constraint Handling
%%          in Prolog)
%%
%%     CHIP is copyrighted by ECRC GmbH and ICL (c) 1989
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================
csp_solve(Objects, Constraints) :-

    get_expanded_list_Constrs(Constraints,Objects,ExpCL),

    apply_constraints_on_object(Objects,VARS),

    write('++++++++++++++++++++++++++++'),nl,
    write('CURRENT SET OF CONSTRAINTS ARE'),nl,nl,s_write(VARS),
    nl, write('++++++++++++++++++++++++++++'),nl,

    apply_constraints(ExpCL),

    write('++++++++++++++++++++++++++++'),nl,
    write('FINAL SET OF CONSTRAINTS ARE'),nl,nl,s_write(VARS),
    nl, write('++++++++++++++++++++++++++++'),nl,

    Solution = Objects.

csp_solve(Objects, Constraints) :-
    nl,nl,nl,nl,
```

```
    write('Cant Generate a Legal Solution for Constraints'),nl,
    write('Try a *bigger* grid'),nl, fail.

%-----------------------------------------------------
%     apply_constraints_on_object(Objects,VARS)
%
%       Apply the constraints to the objects,
%        1) their domain
%        2) size
%-----------------------------------------------------
apply_constraints_on_object(Objects,VARS) :-
    apply_constraints_on_object(Objects,[],VARS), !.
apply_constraints_on_object([],VARS,VARS).
apply_constraints_on_object([Obj1|RemObjs],CurrVars,VARS) :-
    get_vars_from_object(Obj1,_ID,Tag,L,R,B,T),
    apply_constrs_now(ID,Tag,L,R,B,T),
    NewVars = [L,R,B,T| CurrVars],
    apply_constraints_on_object(RemObjs,NewVars,VARS).


apply_constrs_now(_ID,Tag,L,R,B,T) :-
    !,
    max_grid_size_x(MaxX),
    max_grid_size_y(MaxY),
    L::0..MaxX,
    R::0..MaxX,
    B::0..MaxY,
    T::0..MaxY,
    bil_get_icon_data(Tag,_NickName,SizeX,SizeY, IconData),
    R #= L + SizeX,
    T #= B + SizeY.



%-----------------------------------------------------
%  set_CHIP_domains(VARS)
%
%     set a domain for each variable
%-----------------------------------------------------
set_CHIP_domains([]).
set_CHIP_domains([Var1|RemVars]) :-
    write('VAR is '),write(Var1), nl,
    Var1::0..100,
    set_CHIP_domains(RemVars).

%-----------------------------------------------------
%  apply_constraints(Constraints)
```

```
%
%     make the calls to the primitive set,
%     one constraint at a time
%-----------------------------------------------------------
apply_constraints([]) :- nl, nl, write('FINISHED CONSTRAINTS').
apply_constraints([Constr1|RemConstrs]) :-
    nl,nl,write('Constr is '), write(Constr1),
    activate_constraint(Constr1),
    nl,write('NOW the Constr is '), write(Constr1),
    apply_constraints(RemConstrs), !.

activate_constraint(NotConstr) :-
     NotConstr =.. [not,Constr1], !,
     Constr1 =.. [Rel,Obj1,Obj2,P],
     activate_not_constraint(Rel,Obj1,Obj2,P).

activate_constraint(Constr1) :-
     call(Constr1), !.

activate_not_constraint(right,Obj1,Obj2,P) :- call(not_right(Obj1,Obj2,P)).
activate_not_constraint(left, Obj1,Obj2,P) :- call(not_left(Obj1, Obj2,P)).
activate_not_constraint(above,Obj1,Obj2,P) :- call(not_above(Obj1,Obj2,P)).
activate_not_constraint(below,Obj1,Obj2,P) :- call(not_below(Obj1,Obj2,P)).
```

Appendix I

Graphics Routines

```
%%=============================================
%% loader
%%
%%      Loads the files for the Chi-to-Picture Program
%%
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================

%---------------------------------------------
% Load standard Prolog Routines
%---------------------------------------------

:- consult('prolog.ini').                    % standard prolog init routines
:- consult('prologa.ini').                   % arity compatible prolog routines

%---------------------------------------------
% Load the Graphics Routines
%---------------------------------------------

?- use_module(library(gmlib)).

:- consult('make-lf-window.gm').             % Each LF window graphics module
:- consult('make-main-window.gm').           % Main TTPS window graphics module
:- consult('plot-spatial-objects.gm').       % Plot Spatial Objects module
:- consult('plot-temporal-objects.gm').      % Plot Temporal Objects module


:- consult('icon.lib').                      % The Icon Library

%---------------------------------------------
% Top Level of TTPS
%---------------------------------------------
```

```
:- consult('chip2pic.sys').          % Top level of C2P
```

```
%%=============================================
%% csp2pic.sys
%%
%%      Top Level of CSP 2(TO)  Picture System
%%
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================

draw(FileName) :-

%
%   Read in the Chip output (object positions)
%
    concat('INPUT/', FileName, FILENAME),
    open(FILENAME,read,S),
    read(S,SolutionInfo),
    close(S), !,

%
%   Initialize the XWIP graphics and build icons
%

%     use_module(library(gmlib)),
    start,

    SolutionInfo =.. [solutionInfo,Sentence,Pictures],

%
%   Draw the pictures
%
    draw_next_picture(Pictures), !,

    build_ttps_main_window(Sentence,Pictures), !,

%
%   Finished
%
    write('All LFs have been displayed for this sentence.'),
    end.  % to terminate GM process

%---------------------------------------------
%   draw each picture pictures
%---------------------------------------------
draw_next_picture([]).
draw_next_picture([FirstPic|RemPic]) :-
     FirstPic =.. [pr_sol,NumCurrLF,PRInfo],
```

```
nl, write('Graphing Solution for Picture #'),
write(NumCurrLF),write('...'), nl,
sun_gm_graphics(NumCurrLF,PRInfo), !,
draw_next_picture(RemPic).
```

```
%%=============================================
%% make-main-window.gm
%%      Graphics Module for the SUN/Sparc! Monochrome
%%      Using GM graphics in SICSTUS
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================

build_ttps_main window(Sentence,PictureSols) :-

        count_LFs(PictureSols,NumLFs),
        get_set_of_objects(PictureSols,NumObjs,ObjSet),

        WinWidthEstimate is NumObjs * 80,
        (WinWidthEstimate < 300 -> WinWidth = 300;
         WinWidth = WinWidthEstimate),

        V <= view(WinWidth,180),
        H = hbox([space,button("Quit",quit,font("12x24")),space]),
        B <= vbox([V,border,space(H)]),
        W <= window("Text To Pictures System",B),

        V => string(20,160,"The Sentence is:"),
        V => setfont("*-times-bold-r-*-140-*"),
        V => string(50,140,Sentence),
        V => setfont("*-times-bold-r-12-120-*"),
        V => string(20,115,"The Number of LFs is"),
        V => setfont("*-times-bold-r-*-140-*"),
        V => string(145,114,NumLFs),
        V => setfont("*-times-bold-r-12-120-*"),
        V => string(20,100,"The Number of Pictorial Objects is"),
        V => setfont("*-times-bold-r-*-140-*"),
        V => string(230,99,NumObjs),
        V => setfont("*-times-bold-r-12-120-*"),

        draw_icon_defintions(NumObjs,ObjSet,WinWidth,V),

        V => enable,
        W => open,

        repeat,
        waitevent(E),
        E = button(_,quit),
        W => close, !.


%---------------------------------------------
```

```
%   count number of LFs
%-----------------------------------------------------
count_LFs(Pictures,FinalCount) :-
    count_LFs(Pictures,0,FinalCount).
count_LFs([],FinalCount,FinalCount).
count_LFs([_FirstPic|RemPics],OldCount,FinalCount) :-
    NewCount is OldCount + 1,
    count_LFs(RemPics,NewCount,FinalCount).

%-----------------------------------------------------
%   get_set_of_objects
%-----------------------------------------------------
get_set_of_objects(PictureSols,NumObjs,ObjSet) :-
    get_soo_each_picture(PictureSols,0,[],NumObjs,ObjSetR),
    reverse(ObjSetR,ObjSet).

get_soo_each_picture([],Objs,Set,Objs,Set).

get_soo_each_picture([Pic_DblWin|RemPic],CurrNumObjs,CurrObjSet,
                                                TotObjs,TotSet) :-
    Pic_DblWin =.. [pr_sol,_NumCurrLF,[1,_WT1,_TimeInfo1,SpatialInfo1,
                                        2,_WT2,_TimeInfo2,SpatialInfo2]],
    build_unique_list(SpatialInfo1,CurrObjSet,CurrNumObjs,Int1Set,Int1Objs),
    build_unique_list(SpatialInfo2,Int1Set,Int1Objs,Int2Set,Int2Objs),
    get_soo_each_picture(RemPic,Int2Objs,Int2Set,TotObjs,TotSet).

get_soo_each_picture([Pic_SglWin|RemPic],CurrNumObjs,CurrObjSet,
                                                TotObjs,TotSet) :-
    Pic_SglWin =.. [pr_sol,_NumCurrLF,[_WinNum,_WT,_TimeInfo,SpatialInfo]],
    build_unique_list(SpatialInfo,CurrObjSet,CurrNumObjs,IntSet,IntObjs),
    get_soo_each_picture(RemPic,IntObjs,IntSet,TotObjs,TotSet).


build_unique_list([],TotalList,TotalCount,TotalList,TotalCount).
build_unique_list([Obj1|RemObjs],CurrList,CurrCount,TotalList,TotalCount) :-
  is_object_unique(Obj1,CurrList,CurrCount,NewList,NewCount),
  build_unique_list(RemObjs,NewList,NewCount,TotalList,TotalCount).


is_object_unique(Obj1,CurrList,CurrCount,NewList,NewCount) :-
  unique_check(Obj1,CurrList,Flag),
  (Flag -> NewList = [Obj1|CurrList], NewCount is CurrCount + 1;
           NewList = CurrList, NewCount = CurrCount).


unique_check(_Obj1,[], true).                          %is unique
%unique_check(_Obj1,[], false).                         %or terminate
```

```
unique_check(Obj1,[CObj1|_CRemObjs],Flag) :-
    obj_member_checkAgree(Obj1,CObj1), !, Flag = false.
unique_check(Obj1,[_CObj1|CRemObjs],Flag) :-
    unique_check(Obj1,CRemObjs,Flag).


obj_member_checkAgree(Obj1,Obj2) :-
   Obj1 = pictorial_object([_ObjID1,Word,_,_,_,_]),
   Obj2 = pictorial_object([_ObjID2,Word,_,_,_,_]).


%NumObjs = 3,
%ObjSet = [
%2          pictorial_object([obj7,alan,7,28,0,42]),
%          [obj3,say_StateThat,28,58,25,45],
%          [obj1,proposition,185,225,0,30]
%          ].

%count_objs(Solution,FinalCount) :-
%    count_objs(Solution,0,FinalCount).
%count_objs([],FinalCount,FinalCount).
%count_objs([_FirstPic|RemObjs],OldCount,FinalCount) :-
%    NewCount is OldCount + 1,
%    count_objs(RemObjs,NewCount,FinalCount).

%-----------------------------------------------
%    draw the icon defintions
%-----------------------------------------------
draw_icon_defintions(NumObjs,ObjSet,WinWidth,V) :-

    DrawPos is NumObjs + 0.2,
    ObjDist1 is WinWidth / DrawPos,
    ObjDist is ObjDist1 // 1,
    draw_each_icon_defn(ObjSet,ObjDist,V),
    PDC is WinWidth // 2,
    PDX is PDC - 60,
    V => string(PDX,0,"PICTORIAL DEFINITIONS").

draw_each_icon_defn(ObjSet,ObjDist,V) :-
    StartPoint1 is ObjDist // 2,
    StartPoint is 0 - StartPoint1,
    draw_each_icon_defn(ObjSet,ObjDist,StartPoint,V).

draw_each_icon_defn([],_ObjDist,_LastPos,_V).
draw_each_icon_defn([Obj1|RemObjs],ObjDist,LastPos,V) :-
    Obj1 = pictorial_object([_ObjID,ObjWord,_L,_R,_B,_T]),
    CtrObj is LastPos + ObjDist,
```

```
    check_if_rv(ObjWord,ObjWordNotRV),

    bil_get_icon_data(ObjWordNotRV,ObjNickName,IconSizeX,_IconSizeY,IconFile),
    string_length(ObjNickName,WordLength),
    WordOffset1 is WordLength * 3.1,
    WordOffset is WordOffset1 // 1,

    WordPos is CtrObj - WordOffset,
    HalfIconSizeX is IconSizeX // 2,
    IconPos is CtrObj - HalfIconSizeX,

    V => string(WordPos,25,ObjNickName),

    V => bitmap(IconPos, 40,IconFile),

    draw_each_icon_defn(RemObjs,ObjDist,CtrObj,V).


check_if_rv(ObjWord,ObjWordNotRV) :-
    string_search(_C,'_Obj',ObjWord,Loc1), !,
    string_length(ObjWord,Len1),
    ( Loc1 = Len1 -> ObjWordNotRV = ObjWord;
      name(ObjWord,ObjWordL),
      name('_Obj',ObjL),
      append(ObjWordNotRVL,ObjL,ObjWordL),
      name(ObjWordNotRV,ObjWordNotRVL) ).
```

```
%%==========================================
%% make-lf-window.gm
%%      Graphics Module for the SUN/Sparc! Monochrome
%%      Using GM graphics in SICSTUS
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%==========================================


sun_gm_graphics(LF_Number,PRInfo) :-
        decide_window_style(LF_Number,PRInfo).

decide_window_style(LF_Number,PRInfo) :-
       (PRInfo = [1,WT1,T1,S1,2,WT2,T2,S2] ->
           make_double_prw(LF_Number,WT1,T1,S1,WT2,T2,S2); true),
       (PRInfo = [1,WT1,T1,S1] ->
           make_single_prw(LF_Number,1,WT1,T1,S1); true).

%------------------------------------------
%  Single PRW    One Spatial and One Temporal Window
%------------------------------------------

make_single_prw(LF_Number, _Win_Number, _WinTitle,
                              Temporal_Info, Spatial_Info) :-

        Spatial_View <= view(300,160),
        Temporal_View <= view(300,50),

        B <= vbox([frame(Spatial_View),space(frame(Temporal_View))]),

        string_concat('Pictorial Representation ',LF_Number,WinName),
        W <= window(WinName,B),

        plot_spatial_objects(Spatial_Info, Spatial_View),
        plot_temporal_objects(Temporal_Info, Temporal_View),

        Spatial_View => enable,
        Temporal_View => enable,

        W => open.

%------------------------------------------
%  Double PRW    Two Spatial and Two Temporal Window
%------------------------------------------
make_double_prw(LF_Number, WinTitle1,Temporal_Info1, Spatial_Info1,
                              WinTitle2,Temporal_Info2, Spatial_Info2) :-
```

```
%       Spatial_View1 <= view(300,110),
        Spatial_View1 <= view(300,150),
        Temporal_View1 <= view(300,50),
%       Spatial_View2 <= view(260,110),
        Spatial_View2 <= view(260,150),
        Temporal_View2 <= view(300,50),

        B <= vbox([frame(Spatial_View1),space(frame(Temporal_View1)),
                   border,border,border,border,border,border,
                   frame(Spatial_View2),
                   space(frame(Temporal_View2))]),

        string_concat('Pictorial Representation ',LF,Number,WinName),
        W <= window(WinName,B),

        Spatial_View1 => setfont("5x8"),
        Spatial_View1 => string(5,140,WinTitle1),

        Spatial_View1 => setfont("5x8"),
        Spatial_View1 => string(290,140,"PRW1"),

        Temporal_View1 => setfont("5x8"),
        Temporal_View1 => string(275,40,"PRW2"),

        plot_spatial_objects(Spatial_Info1, Spatial_View1),
        plot_temporal_objects(Temporal_Info1, Temporal_View1),

        Spatial_View2 => setfont("5x8"),
        Spatial_View2 => string(5,140,WinTitle2),

        Spatial_View2 => setfont("5x8"),
        Spatial_View2 => string(290,140,"PRW3"),

        Temporal_View2 => setfont("5x8"),
        Temporal_View2 => string(275,40,"PRW4"),

        plot_spatial_objects(Spatial_Info2, Spatial_View2),
        plot_temporal_objects(Temporal_Info2, Temporal_View2),

        Spatial_View1  => enable,
        Temporal_View1 => enable,
        Spatial_View2  => enable,
        Temporal_View2 => enable,

        W => open.
```

```
%%=============================================
%% plot-spatial-objects.gm
%%
%%    Plot the Spatial Objects
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================

% Constants defined
%
mGridX(300).
mGridY(150).
win_0X(10).
win_0Y(10).

%---------------------------------------------
%
% Plot the Objects
%

plot_spatial_objects(Object_List, GMV) :-
    gr_determine_grid_size(Object_List,ShiftLeft,ShiftDown,SpreadX,SpreadY),
    gr1_draw_objects(        Object_List,ShiftLeft,ShiftDown,SpreadX,SpreadY,
                               GMV),    !.



%---------------------------------------------
%
% set up a grid size
%
gr_determine_grid_size(_Object_List,0,0,260,120).

gr_determine_grid_size(Object_List,ShiftLeft,ShiftDown,SpreadX,SpreadY) :-
    gr_dgs(Object_List,0,0,0,0,LeastX,LeastY,MaxX,MaxY),
    ShiftLeft = LeastX,
    ShiftDown = LeastY,
    gr_set_spread(LeastX,MaxX,SpreadX),
    gr_set_spread(LeastY,MaxY,SpreadY), !.

gr_set_spread(Least,Max,Spread) :-
    S1 is Max - Least,
    S1 > 5, !,
    Spread = S1.
gr_set_spread(_Least,_Max,5).

gr_dgs([],LX,LY,MX,MY,LX,LY,MX,MY).
```

```
gr_dgs([pictorial_object([_P,_D,X,UX,Y,UY])|RemVOL],
            OLX,OLY,OMX,OMY,NLX,NLY,NMX,NMY) :-
    gr_dgs_lt(X, OLX,ILX),
    gr_dgs_lt(Y, OLY,ILY),
    gr_dgs_gt(UX,OMX,IMX),
    gr_dgs_gt(UY,OMY,IMY),
    gr_dgs(RemVOL,ILX,ILY,IMX,IMY,NLX,NLY,NMX,NMY).

gr_dgs_lt(T,Old,New) :-
    T < Old, !, New = T.
gr_dgs_lt(_T,Old,Old).
gr_dgs_gt(T,Old,New) :-
    T > Old, !, New = T.
gr_dgs_gt(_T,Old,Old).
%-----------------------------------------------------------


%-----------------------------------------------------------
% draw the objects
%
%


gr1_draw_objects([],_SL, SD,_MX, MY, _GMV).
gr1_draw_objects([H|T],SL,SD,MX,MY, GMV) :-
    gr1_draw_object(H,  SL,SD,MX,MY, GMV),
    gr1_draw_objects(T, SL,SD,MX,MY, GMV).

gr1_draw_object(pictorial_object([_P,Data,X,_UX,Y,_UY]),
            ShiftLeft, ShiftDown, MaxX, MaxY, GMV):-
%%    gr_scale(X,Y,ShiftLeft, ShiftDown, MaxX, MaxY, NewX, NewY),
    gr_scale(X,Y,ShiftLeft, ShiftDown, MaxX, MaxY, NewX, NewY,Data),
    gr_draw_icon(Data,NewX,NewY, GMV).


gr_scale(X, Y, ShiftLeft, ShiftDown, _MaxX, _MaxY, NewX, NewY, _IPtr) :-

% get visual gird sizes and zero locations
%
%    mGridX(_MGridX),mGridY(MGridY),

    win_0X(Win_0X),win_0Y(Win_0Y),

% adjust out X,Y offset to a new (0,0) location, because of
% negative X and Y values
%
    X_Offset        is X  - ShiftLeft,
    Y_Offset        is Y  - ShiftDown,
```

```
% scale to full size of screen
% (for now set to 1.0 because of also sizing icons would be needed)
%
%%    X_Scale_factor  is MGridX / MaxX,
%%    Y_Scale_factor  is MGridY / MaxY,
    X_Scale_factor  is 1,
    Y_Scale_factor  is 1,

    X_Grid          is X_Offset * X_Scale_factor,
    Y_Grid          is Y_Offset * Y_Scale_factor,

    NewX             is X_Grid + Win_0X,
    NewY             is Y_Grid + Win_0Y.

%-----------------------------------------------

%-----------------------------------------------
% draw the icon
%
%

gr_draw_icon(IconTag,X,Y, GMV) :-
        bil_get_icon_data(IconTag,_ObjNickName,_IconSizeX,_IconSizeY,IconFile),
        bitmap_file(IconFile,BMF),

        GMV => bitmap(X,Y,BMF).

% Save the path name so the bitmap files can later be found.

:- absolute_file_name('.', PATH), assert(gm_path(PATH)).

bitmap_file(BF, AFN) :- atom(BF), name(BF, BFS), bitmap_file(BFS, AFN).
bitmap_file(BFS, AFN) :-
        BFS1 = [47|BFS],
        gm_path(PATH),
        name(PATH,PS),
        append(PS, BFS1, AFN).
```

```
%%=============================================
%% plot-temporal_objects.gm
%%
%%    Plot the Temporal Objects
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================

%---------------------------------------------
%
% Plot the Objects
%

plot_temporal_objects(TemporalInfo, Temporal_V) :-
%    nl,write(TemporalInfo),
    apply_temporal_constraints(TemporalInfo,Temporal_V).


apply_temporal_constraints([], _Temporal_V).
apply_temporal_constraints([CS1|CSS],Temporal_V) :-
    CS1 =.. [Func,temporal,Arg1,Arg2],
    CS2 =.. [Func,Temporal_V,Arg1,Arg2],
    call(CS2),
    apply_temporal_constraints(CSS, Temporal_V).

%---------------------------------------------
%  PAST-NOW-FUTURE   Scale and Time Plotting Routine
%---------------------------------------------
% New version only with button
%
plot_time_and_scale2(Temporal_V,Time,past_now_future) :-
%
% set points of reference

    TLineIconX = 0,
    TLineIconY = 5,

%    LengthOfTime    = 260,
    TimeZeroY       is TLineIconY + 2,
    TimeZeroX       is TLineIconX + 0,

    TimeBeginRef    =  TimeZeroX,
    TimePastRef     is TimeBeginRef + 14,
    TimeNowRef      is TimeBeginRef + 131,
    TimeFutureRef   is TimeBeginRef + 247,

    HalfWidth_TMarker = 10,
```

```
    (Time = 'past'   -> TMarkerX is TimePastRef   - HalfWidth_TMarker; true),
    (Time = 'now'    -> TMarkerX is TimeNowRef    - HalfWidth_TMarker; true),
    (Time = 'future' -> TMarkerX is TimeFutureRef - HalfWidth_TMarker; true),
    TMarkerY = TLineIconY,
    Temporal_V => bitmap(TMarkerX,TMarkerY,"ICONS/oval.bit"),


     Temporal_V  => setfont("5x8"),
     Temporal_V => string(TimePastRef,TimeZeroY,"PAST"),
     Temporal_V => string(TimeNowRef,TimeZeroY,"NOW"),
     Temporal_V => string(TimeFutureRef,TimeZeroY,"FUTURE").



%-----------------------------------------------
%  PAST-NOW-FUTURE   Scale and Time Plotting Routine
%-----------------------------------------------
% Orginal version with *SCALE* and *ARROWS*
%
plot_time_and_scale(Temporal_V,Time,past_now_future) :-

%
% set points of reference
% this is based upon the p-n-f timeline icon of 5x260 pixels

    TLineIconX = 20,
    TLineIconY = 12,

%   LengthOfTime    = 260,
    TimeZeroY      is TLineIconY + 3,
    TimeZeroX      is TLineIconX + 0,

    TimeBeginRef    =  TimeZeroX,
    TimePastRef     is TimeBeginRef + 14,
    TimeNowRef       is TimeBeginRef + 131,
    TimeFutureRef   is TimeBeginRef + 247,

%
% draw the time scale display
%
    Temporal_V => string(25,0,"Past"),
    Temporal_V => string(140,0,"Now"),
    Temporal_V => string(250,0,"Future"),
    Temporal_V => bitmap(TLineIconX,TLineIconY,"ICONS/timeline.bit"),

%
% place the time is marker
```

```
%
   HalfWidth_TMarker = 5,
   (Time = 'past'   -> TMarkerX is TimePastRef   - HalfWidth_TMarker; true),
   (Time = 'now'    -> TMarkerX is TimeNowRef    - HalfWidth_TMarker; true),
   (Time = 'future' -> TMarkerX is TimeFutureRef - HalfWidth_TMarker; true),
   TMarkerY = TimeZeroY,
   Temporal_V => bitmap(TMarkerX,TMarkerY,"ICONS/timeisDownArrow.bit").


%-----------------------------------------------
%  MONTHLY  Scale and Time Plotting Routine
%-----------------------------------------------
plot_time_and_scale(Temporal_V,[Year,Month,Day],days_of_month) :-

%
% set points of reference
% this is based upon the month timeline icon of 5x240 pixels

   TLineIconX = 30,
   TLineIconY = 30,

   LengthOfTimeScale   = 240,
   LengthOfDay is LengthOfTimeScale // 30,

   TimeZeroY      is TLineIconY + 0,
   TimeZeroX      is TLineIconX + 0,

   DateFromOne is Day - 1,
   DayMark is DateFromOne * LengthOfDay,

%
% draw the time scale display
%
   Temporal_V => string(28,33,"1"),
   Temporal_V => string(59,33,"5"),
   Temporal_V => string(96,33,"10"),
   Temporal_V => string(138,33,"15"),
   Temporal_V => string(178,33,"20"),
   Temporal_V => string(218,33,"25"),
   Temporal_V => string(266,33,"31"),
   Temporal_V => bitmap(TLineIconX,TLineIconY,"ICONS/month.bit"),

%
% place the time is marker
%
   HalfWidth_TMarker = 4,
   FullHeight_TMarker = 5,
   TMarkerX1 is TimeZeroX + DayMark,
```

```
    TMarkerX  is TMarkerX1 - HalfWidth_TMarker,
    TMarkerY is TimeZeroY - FullHeight_TMarker,
    Temporal_V => bitmap(TMarkerX,TMarkerY,"ICONS/timeisUpArrow.bit").

%-----------------------------------------------
%  DAYS OF WEEK  Scale and Time Plotting Routine
%-----------------------------------------------
plot_time_and_scale(Temporal_V,[Year,Month,Day],days_of_week) :-

%
% set points of reference
% this is based upon the month timeline icon of 5x240 pixels

    TLineIconX = 30,
    TLineIconY = 30,

    LengthOfTimeScale    = 240,
    LengthOfDay is LengthOfTimeScale // 6,

    TimeZeroY      is TLineIconY + 0,
    TimeZeroX      is TLineIconX + 0,

     DateFromSun0 is Day mod 7,
     DateFromSun1 is DateFromSun0 + 2,
     DateFromSun  is DateFromSun1 mod 7,

     DayMark is DateFromSun * LengthOfDay,

%
% draw the time scale display
%
    Temporal_V => string(23,33,"Sun"),
    Temporal_V => string(63,33,"Mon"),
    Temporal_V => string(104,33,"Tue"),
    Temporal_V => string(144,33,"Wed"),
    Temporal_V => string(183,33,"Thu"),
    Temporal_V => string(222,33,"Fri"),
    Temporal_V => string(263,33,"Sat"),
    Temporal_V => bitmap(TLineIconX,TLineIconY,"ICONS/week.bit"),

%
% place the time is marker
%
    HalfWidth_TMarker = 4,
    FullHeight_TMarker = 5,
    TMarkerX1 is TimeZeroX + DayMark,
    TMarkerX  is TMarkerX1 - HalfWidth_TMarker,
    TMarkerY is TimeZeroY - FullHeight_TMarker,
```

```
Temporal_V => bitmap(TMarkerX,TMarkerY,"ICONS/timeisUpArrow.bit").
```

Appendix J

Icon Library

```
%%=============================================
%% icon.lib
%%
%%    This is Icon Library
%%
%%
%%  (c) 1992 N. D. Ludlow, University of Edinburgh
%%=============================================

%-------------------------------------------------
%  bil_get_icon_data(Tag,NickName,SizeX,SizeY,IconFile)
%
%      Tag       -- the word used in Clare
%      NickName -- the word used on pictorial defn menu
%      SizeX     -- the size of icon horizontally
%      SizeY     -- the size of icon vertically
%      IconFile -- the file name of X bitmap
%
%    *Note: files with RV extension are reverse video.
%    GM lib does not have an automatic feature of reverse video
%    of icons, so reverse video icons are prepared ahead of time.
%    When SICStus GMlib is improved, that could be removed.
%-------------------------------------------------

bil_get_icon_data('alan','Alan',21,42,'ICONS/alan.bit').

bil_get_icon_data('car_WheeledVehicle','Car',50,25,'ICONS/car.bit').
bil_get_icon_data('car_WheeledVehicle_Obj','Car',50,25,'ICONS/carRV.bit').
bil_get_icon_data('carla','Carla',19,40,'ICONS/carla.bit').
bil_get_icon_data('cat_Animal','Cat',18,22,'ICONS/cat.bit').
bil_get_icon_data('cat_Animal_Obj','Cat',18,22,'ICONS/catRV.bit').

bil_get_icon_data('drive_CauseToOperate','Drive',19,19,'ICONS/drive.bit').
```

```
bil_get_icon_data('flower_Obj','Flower',20,40,'ICONS/flower.bit').

bil_get_icon_data('ground','Ground',260,1,'ICONS/ground.bit').

bil_get_icon_data('he_MalePerson','He',21,42,'ICONS/man.bit').
bil_get_icon_data('hill_HighGround','Hill',65,50,'ICONS/hill.bit').

bil_get_icon_data('machine_Device','Telescope',7,30,'ICONS/machine.bit').
bil_get_icon_data('machine_Device_Instr','Tscope in use',
                                                38,7,'ICONS/machineI.bit').
bil_get_icon_data('man_MalePerson','Man',21,42,'ICONS/man.bit').
bil_get_icon_data('man_Masculine','Man',21,42,'ICONS/man.bit').
bil_get_icon_data('man_MalePerson_Obj','Man',21,42,'ICONS/manRV.bit').
bil_get_icon_data('man_Masculine_Obj','Man',21,42,'ICONS/manRV.bit').

bil_get_icon_data('pot','Flower Pot',30,20,'ICONS/pot.bit').
bil_get_icon_data('proposition','Event 2',26,16,'ICONS/proposition.bit').

bil_get_icon_data('say_StateThat','Say',40,30,'ICONS/say.bit').
bil_get_icon_data('see_LookAt','See',19,21,'ICONS/see.bit').
bil_get_icon_data('speaker','I',15,30,'ICONS/speaker.bit').

bil_get_icon_data('table_Furniture','Table',30,30,'ICONS/table.bit').
bil_get_icon_data('telescope1','Telescope',7,30,'ICONS/telescope.bit').
bil_get_icon_data('telescope1_Instr','Tscope in use',
                                              38,7,'ICONS/telescopeI.bit').

bil_get_icon_data('unknown','Who knows?',16,16,'ICONS/unknown.bit').

bil_get_icon_data('woman_FemalePerson','Woman',19,40,'ICONS/woman.bit').
bil_get_icon_data('woman_FemalePerson_Obj','Woman',19,40,'ICONS/womanRV.bit').

bil_get_icon_data(_Dummy,Width,Height,PL) :-
bil_get_icon_data('unknown',Width,Height,PL).
```

Bibliography

[Allen 83] James F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(1):832–843, 1983.

[Alshawi & van Eijck 89] H. Alshawi and J. van Eijck. Logical Forms in the Core Language Engine. In *Proceedings of the 27th Annual Meeting of the ACL*, pages 26–29, Vancouver, Canada, June 1989.

[Alshawi 90] H. Alshawi. Resolving Quasi Logical Forms. *Association of Computational Linguistics*, Fall 1990.

[Alshawi *et al* 88] H. Alshawi, J. van Eijck, R. C. Moore, D. B. Moran, F. C. N. Pereira, A. G. Smith, and S. G. Pulman. Interim report on the sri core language engine. Technical Report CCSRC-005, SRI International / Cambridge Computer Science Research Center, July 1988.

[Alshawi *et al* 91] H. Alshawi, Dave Carter, Richard Crouch, Steve Pulman, Manny Rayner, and Arnold Smith. Clare-2 software manual. Software Manual SRI Project 8468, SRI International Cambridge Computer Science Research Centre, November 1991.

[Aurnague & Borillo 90a] Michel Aurnague and Mario Borillo. A Formal Semantics for Internal Localization: An Essay on Spatial Commonsense Knowledge. In *AIMSA 90*, Varna, Bulgaria, 1990.

[Aurnague & Borillo 90b] Michel Aurnague and Mario Borillo. Semantics of Internal Localization Names in French. Seminar, 2nd European Summer School in Language, Logic and Information, August 1990.

[Barlow 90] Horace Barlow. What does the brain see? How does it understand? In Horace Barlow, Colin Blakemore, and Miranda Weston-Smith, editors, *Images and Understanding*, Cambridge, England, 1990. Cambridge University Press.

[Bodington & Elleby 88] Rob Bodington and Peter Elleby. Justification and Assumption–Based Truth Maintenance Systems: When

and How to use them for Constraint Satisfaction. In *in Proceedings of AISB WOrkshop on Reason Maintenance Systems and their Applications*, Leeds, England, April 1988. Ellis Harwood.

[Borillo & Borillo 90] Mario Borillo and Andree Borillo. Seminar on Spatial Expressions. Course Notes, 2nd European Summer School in Language, Logic and Information, August 1990.

[Borning *et al* 89] Alan Borning, Michael Maher, Amy Martindale, and Molly Wilson. Constraint Hierarchies and Logic Programming. *Association of Computational Linguistics*, 1989.

[Burrow 55] Thomas Burrow. *The Sanskrit language.* Faber and Faber, London, 1955.

[Carlsson 91a] Mats Carlsson. Sicstus prolog library manual. Software Manual T91:12B, Swedish Institute of Computer Science, October 1991.

[Carlsson 91b] Mats Carlsson. Sicstus prolog user's manual. Software Manual T91:11B, Swedish Institute of Computer Science, October 1991.

[Carter 92] David Carter. VEX–Lexicon Entry in the CLARE system. Poster at the 3rd Applied Natural Lanugage Conference, April 1992.

[Cortes 89] Luis Alberto Pineda Cortes. *GRAFLOG: A Theory of Semantics for Graphics with Applications to Human–Computer Interaction and CAD Systems.* Unpublished PhD thesis, University of Edinburgh, Edinburgh, Scotland, 1989.

[deKleer 86a] Johan de Kleer. An Assumption–based TMS. *Artificial Intelligence*, pages 127–161, 1986.

[deKleer 86b] Johan de Kleer. Problem Solving with the ATMS. *Artificial Intelligence*, pages 197–224, 1986.

[Dincbas *et al* 88a] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. Applications of CHIP to Industrial and Engineering Problems. In *Proceedings of the Fist International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Tullahoma, Tennessee, June 1988.

[Dincbas *et al* 88b] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The Constraint Logic Programming Language CHIP. In *Proceedings of the In-*

ternational Conference on Fifth Generation Computer Systems (FGCS-88), Tokyo, November 1988.

[Dincbas *et al* 88c] M. Dincbas, H. Simonis, and P. Van Hentenryck. Solving the Car–Sequencing Problem in Constraint Logic Programming. In *Proceedings of the 8th European Conference on Artificial Intelligence (ECAI-88)*, pages 290–295, Munich, August 1988.

[Doyle 79] Jon Doyle. A Truth Maintenance System. *Artificial Intelligence*, pages 231–272, 1979.

[Dunden 89] Dunden. *The Oxford–Dunden Pictorial English Dictionary.* Oxford University Press, 1989.

[Feiner & McKeown 90] Steven K. Feiner and Kathleen R. McKeown. Coordinating Text and Graphics in Explanation Generation. In *IEEE Conference*, pages 442–449, 1990.

[Fillmore 68] Charles J. Fillmore. The Case for Case. In E. Bach and R. T. Harms, editors, *Universals in Linguistic Theory*, pages 151–162, Chicago, 1968. Holt, Rinehart and Winston.

[Fillmore 75] Charles J. Fillmore. An alternative to checklist theories of meaning. In *Proceedings of the First Annual Meeting of the Berkeley Linguistics Society*, pages 123–131, Berkeley, California, 1975. University of California.

[Fodor 83] Jerry A. Fodor. *The Modularity of Mind.* MIT Press, Cambridge, Massachusetts, 1983.

[Foley *et al* 90] James Foley, Andries van Dam, Steven Feiner, and John Hughes. *Computer Graphics: Principles and Practice.* Addison Wesley, Reading, Massachusetts, 1990.

[Freeman-Benson & Wilson 89] Bjorn N. Freeman-Benson and Molly Wilson. A General Algorithm for Incremental Satisfaction of Constraint Hierarchies. In *North American Conference on Logic Programming 1990*, Cleveland, Ohio, October 1989.

[Garnham 87] A. Garnham. *Mental Models as representations of discourse and text.* Chichester: Ellis Horwood, 1987.

[Gazdar *et al* 85] Gerald Gazdar, Ewan Klein, Geoffrey Pullum, and Ivan Sag. *Generalized Phrase Structure Grammar.* Basil Blackwell, 1985.

[Golvina 74] T. P. Golvina. Comparative Study of the Development of Comprehension in Weak-Sighted School Children of Normal and Abnormal Intelligence. In *Osobennosti posnavatel'noy deyatel'nosti slepykh i slabovidyashchikh shkol'nikov (Characteristics of the Cognitive Activity*

of Blind and Weak-sighted School children), volume 4, pages 151–162, Leningrad, USSR, 1974.

[Grishman & Hirschman 86] Ralph Grishman and Lynette Hirschman. PROTEUS and PUNDIT: Research in Text Understanding. Technical report, Courant Institute of Mathematical Sciences, New York University, 1986.

[Grishman 86] Ralph Grishman. *Computational Linguistics.* Cambridge University, Cambridge, Massachusetts, 1986.

[Grishman 87] Ralph Grishman. An Introduction to the PROTEUS Parser. Technical report, Courant Institute of Mathematical Sciences, New York University, 1987.

[Grosz 83] Barbara J. Grosz. TEAM: A Transportable Natural Language Interface System. In *Conference on Applied Natural Language Processing*, Santa Monica, California, 1983.

[Hayes 79] P. J. Hayes. The naive physics manifesto. Geneva, Switzerland: Institute of Semantic and Cognitive Studies, 1979.

[Hendrix 77] G. G. Hendrix. The LIFER Manual: A guide to building practical natural language interfaces. Technical Report 138, SRI International, Menlo Park, California, February 1977.

[Hendrix *et al* 78] G. Hendrix, E. Sacerdoti, D. Sagalowiz, and J. SLocum. Developing a Natural Language Interface to Complex Data. *ACM Trans. on Database Systems*, 3(2):105 – 147, 1978.

[Herskovits 86] Annette Herskovits. *Language and spatial cognition: An interdisciplinary study of the prepositions in English.* Cambridge University Press, 1986.

[Hopgood 83] F. R. A. Hopgood. *Introduction to Graphical Kernel System (GKS).* Academic Press, 1983.

[Johnson-Laird 83] P. N. Johnson-Laird. *Mental Models.* Cambridge University Press, Cambridge, 1983.

[Jung 64] Carl G. Jung. *Man and his Symbols.* J. G. Ferguson Publishing, New York, New York, 1964.

[Katz & Fodor 68] Jerrold J. Katz and Jerry A. Fodor. *Using constraints and Structure of language.* Prentice Hall, Englewood Cliffs, NJ, 1968.

[Kielhorn 12] Franz Kielhorn. *A grammar of the Sanskrit language.* Tukarem Javaji, Bombay, 1912.

[Klein 90] Ewan Klein. Problems in Representing Text by Graphics. Espirit Project Seminar, Cognitive Science Department, University of Edinburgh, March 1990.

[Lang *et al* 90] Ewald Lang, Kai Uwe Carstensen, and Geoff Simmons. Modeling Spatial Knowledge on a Linguistic Basis. Technical report, IBM Germany Science Center, Stuttgart, Germany, 1990.

[Ludlow 88] Nelson D. Ludlow. An Automated Conversion of Narrative English Text of Varied Subject Matter to a Structured Data Base. Unpublished M.Sc. thesis, Wright State University, Dayton, Ohio, 1988.

[Ludlow 89] Nelson Ludlow. Graphical Representations of Text: A PhD Thesis Proposal. Discussion Paper No. 81, Department of Artificial Intelligence, University of Edinburgh, August 1989.

[Maienborn 91] Claudia Maienborn. Processing Spatial Knowledge in LILOG. Technical report, IBM Germnay Science Center, Stuttgart, Germany, 1991.

[Mani & Johnson-Laird 82] K. Mani and P. N. Johnson-Laird. The mental representation of spatial descriptions. *Memory and Cognition*, 10:81–87, 1982.

[Marks & Reiter 90] Joseph Marks and Ehud Reiter. Avoiding Unwanted Conversational Implicatures in Text and Graphics. In *IEEE Conference*, pages 450–456, 1990.

[Marr 82] David Marr. *Vision: A computational Investigation into the Human Representation and Processing of Visual Information.* W. H. Freeman, 1982.

[Martin *et al* 83] P. Martin, D. Appelt, and F. Pereira. Transporting and Generality in a Natural-Language Interface System. In *Proceedings 8th International Joint Conference on Artificial Intelligence*, pages 573–581, Karlsruhe, Germany, 1983.

[McCord 82] Michael C. McCord. Using Slots and Modifiers in Logic Grammars for Natural Language. *Artificial Intelligence*, 18:327–367, 1982.

[McCord 87] Michael C. McCord. *Knowledge Systems and Prolog*, chapter 5. Addison Wesley, Reading, Massachusetts, 1987.

[Mel'chuk *et al* 75] Igor A. Mel'chuk, A. P. Ershov, and A. S. Nariniany. RITA–An Experimental Man–Computer System on a Natural Language Basis. In *IJCAI Conference Proceedings*, pages 387–390, Tbilisi, USSR, 1975.

[Mellish 85] Christopher S. Mellish. *Computer Interpretation of Natural Language Descriptions.* Ellis Horwood/Wiley, New York, 1985.

[Meriam-Webster 86] Meriam-Webster. *Webster's Ninth New Collegiate Dictionary.* Collegiate Publishers, 1986.

[Miles 88] Dorothy Miles. *British Sign Language.* BBC Books, London, England, 1988.

[Milne 83] Robert William Milne. *Resolving Lexical Ambiguity in a Deterministic Parser.* Unpublished PhD thesis, University of Edinburgh, Edinburgh, Scotland, UK, 1983.

[Minsky 81] Marvin Minsky. A Framework for Representing Knowledge. In J. Haugeland, editor, *Mind Design*, pages 95–128, Cambridge, Massachuesetts, 1981. MIT Press.

[Montgomery & Neal 91] Christine Montgomery and Jeannette G. Neal. An Evaluation Technique for Natural Language Systems. Seminar, Rome Laboratory, Griffiss AFB, NY, 1991.

[Moore & Swartout 90] Johanna D. Moore and William R. Swartout. Pointing: A Way Toward Explanation Dialogue. In *IEEE Conference*, pages 457–464, 1990.

[Morris & Sagolowiz 77] P. Morris and D. Sagolowiz. Managing network access to a distributed database. In *Proceedings 2nd Berkely Workshop on Distribute Data Management and Computer Networks*, Berkeley, California, May 1977.

[Nishida *et al* 88] Toyoaki Nishida, Atushi Yamada, and Shuji Doshita. Constructing Spatial Images from Natural Language Texts. In *International Sxymposium on Artificial Intelligence: Multimedia Knowledge Processing for Better Human Performance*, August 1988.

[Onyshkevych 88] Boyan A. Onyshkevych. A Survey of the State-of-the-Art of Natural Language Processing in the United States. Technical report, Naval Ocean Systems Center, San Diego, California, August 1988.

[Pain 89] Helen Pain. Ai-1 lecture notes. Lecture notes, Department of Artificial Intelligence, University of Edinburgh, September 1989.

[Palmer *et al* 88] Martha Palmer, Lynette Hirschman, and Deborah Dahl. Text Processing Systems. Tutorial presentation at the 26th Annual Meeting of the Association for Computational Linguistics, 1988.

[Parnwell 91] E. C. Parnwell. *Oxford English Picture Dictionary: Dictionanaire illustre' Anglais-Francais.* Oxford University Press, 1991.

[Pereira & Shieber 87] Fernando C.N. Pereira and Stuart M. Shieber. *Prolog and Natural-Language Analysis.* CSLI Lecture Notes 10. Center for the Study of Language and Information, Stanford, California, 1987.

[Pereira & Warren 80] Fernando C.N. Pereira and D.H.D. Warren. Definite Clause Grammars for Language Analysis. *Artificial Intelligence*, 13:231–278, 1980.

[Pereira 83] Fernando Pereira. Logic for Natural Language Analysis. Technical Report 275, SRI International, 1983.

[Pineda *et al* 88] Luis A. Pineda, Ewan Klein, and John Lee. GRAFLOG: Understanding Drawings through Natural Language. In *6th Annual EUROGRAPHICS Conference*, 1988.

[Pospelov 86] Dimitri Andreivich Pospelov. *Situation Modelling.* Radio Press, Moscow, USSR, 1986.

[Pylyshyn 73] Z. W. Pylyshyn. What the Mind's Eye Tells the Mind's Brain: A critique of Mental Imagery. *Psychological Bulletin*, 80:1–24, 1973.

[Retz-Schmidt 88] Gudula Retz-Schmidt. Various Views on Spatial Prepositions. *AI Magazine*, pages 95–105, Summer 1988.

[Rollins 89] Mark Rollins. *Mental Imagery: On the Limits of Cognitive Science.* Yale University Press, New Haven, 1989.

[Rosch 77] E. Rosch. Human Categorization. In N. Warren, editor, *Advances in cross-cultural psychology*, pages 1–49, London, 1977. Academic Press.

[Sacerdoti 77] E. D. Sacerdoti. Language access to distributed data with error recovery. In *Proceedings 5th International Joint Conference on Artificial Intelligence*, Cambridge, Mass., Aug 1977.

[Sagalowicz 77] D. Sagalowicz. IDA: An intelligent data access program. In *Proceedings 3rd International Conference on Very Large Data Bases*, Tokyo, Japan, Oct 1977.

[Sager 78] Naomi Sager. Natural Language Information Formatting: The Automatic Conversion of Texts to a Structured Data Base. *Advances in Computers*, 17, 1978.

[Sager *et al* 87] Naomi Sager, Carol Friedman, and Margaret Lyman. *Medical Language Processing: Computer Management of Narrative Data.* Addison Wesley, Reading, Massachusetts, 1987.

[Schank 80] Roger C. Schank. Language and Memory. *Cognitive Memory*, 4(3), 1980.

[Scott 92] Donia Scott. Relating Text to Image in Descriptions of Complex Scenes. Seminar, Natural Language Group Seminar, University of Cambridge, February 3rd 1992.

[Senn 84] James A. Senn. *Analysis and Design of Information Systems.* McGraw-Hill, 1984.

[Spark-Jones & Wilks 85] Karen Spark-Jones and Yorick Wilks. *Automatic Natural Language Processing.* Ellis Horwood, 1985.

[Sundheim 91] Beth Sundheim. Initial results from MUC-3 conference. In Jeannette G. Neal and Sharon M. Walter, editors, *Natural Language Processing Systems Workshop.* DARPA/Rome Laboratory, 1991.

[Talmy 75] Leonard Talmy. Semantics and Syntax of Motion. *Syntax and Semantics*, 6, 1975.

[Talmy 78] Leonard Talmy. Figure and Ground in Complex Sentences. In J. Greenberg, C. Ferguson, and E. Moravcsik, editors, *Universals of Human Language*, Stanford, California, 1978. Stanford University Press.

[Talmy 83] Leonard Talmy. How language structures space. In H. Pick and L. Acredolo, editors, *Spatial Orientation: Theory, Research, and Application*, New York, 1983. Plenum Press.

[Talmy 85] Leonard Talmy. Lexicalization patterns: semantic structure in lexical forms. In T. Shopen, editor, *Language typology and syntactic description*, 1985.

[Tufte 90] Edward R. Tufte. *Envisioning Information.* Graphics Press, Cheshire, Connecticut, 1990.

[Vandeloise 86] Claude Vandeloise. The preposition 'in' and the relationship container/contained. *Linguistic Agency University of Duisburg*, Series A(155), 1986.

[Vieu & Borillo 90] Laure Vieu and Mario Borillo. Study of Spatial Expressions: Analysis of "dans". Seminar, 2nd European Summer School in Language, Logic and Information, August 1990.

[Vieu 91] Laure Vieu. *Le preposition dans.* Unpublished PhD thesis, IRIT and Universite de Toulouse, Toulouse, France, 1991.

[Vilian *et al* 86] Marc Vilian, Henry Kautz, and Peter van Beek. Constraint Propogation Algorithms for Temporal Reasoning: A Revised Report. *Proceedings of AAAI-86*, pages 377–382, 1986.

[Wahlster *et al* 91a] W. Wahlster, E. Andre, S. Bandyopadhyay, W. Graf, and T. Rist. WIP: The Coordinated Generation of Multimodal Presentations from a Common Representation. In *Computational Theories of Communication and their Applications*, Berlin, 1991. Springer-Verlag.

[Wahlster *et al* 91b] Wolfgang Wahlster, Elisbeth Andre, Winfried Graf, and Thomas Rist. Designing Illustrated Texts: How language production is influenced by Graphics Generation. In *5th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 8–14, Berlin, 1991.

[Wainer & Thissen 81] Howard Wainer and David Thissen. Graphical Data Analysis. *Annual Review of Psychology*, 32:191–241, 1981.

[Watt 89] Alan H. Watt. *Fundamentals of Three-Dimensional Computer Graphics*. Addison Wesley, Wokingham, England, 1989.

[Wazinski 92] Peter Wazinski. Generating Spatial Descriptions for Cross-modal References. In *3rd Applied Natural Language Conference*, pages 56–63, Trento, Italy, April 1992.

[Wilson 89] Molly Wilson. Extending Hierarchical Constraint Logic Programming: Nonmonotonicity and Inter-Hierarchy Comparison. In *North American Conference on Logic Programming 1990*, Cleveland, Ohio, October 1989.

[Winograd 83] Terry Winograd. *Language as a Cognitive Process. Volume 1, Syntax.* Addison Wesley, Reading, Massachusetts, 1983.

[Woods 73] W. A. Woods. Progress in natural language understanding: An application to lunar geology. In *AFIPS Conference Proceedings 42*, pages 441–450, 1973.

[Woods 78] W. A. Woods. Semantics and Quantification in Natural Language Question Answering. *Advances in Computers*, 17, 1978.

[Yamada *et al* 88] Atushi Yamada, Toyoaki Nishida, and Shuji Doshita. Figuring out Most Plausible Interpretation from Spatial Descriptions. In *Proceedings of the 12th International Conference on Computational Linguistics/COLING88*, Budapest, Hungary, 1988.

[Yamada *et al* 92] Atushi Yamada, Tadashi Yamamoto, Hisashi Ikeda, Toyoaki Nishida, and Shuji Doshita. The Use of Reconstructed Spatial Image in Natural Language Understanding Process. In *Symposium on Reasoning with Diagrammatic Representations*, Stanford University, California, March 1992.